Tvorba mobilní aplikace Relays

Filip Vabroušek

Bakalářská práce
2022

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
Ústav informatiky a umělé inteligence

Akademický rok: 2021/2022

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Filip Vabroušek**
Osobní číslo:	**A19119**
Studijní program:	**B3902 Inženýrská informatika**
Studijní obor:	**Softwarové inženýrství**
Forma studia:	**Prezenční**
Téma práce:	**Tvorba mobilní aplikace Relays**
Téma práce anglicky:	**Creation of the Relays Mobile Application**

Zásady pro vypracování

1. Stručně popište možnosti vývoje mobilní aplikace pro platformy Android, iOS a pro web.
2. Vyberte si vhodné technologie pro vývoj na výše uvedených platformách, včetně technologie serverové strany a stručně je popište.
3. V rámci praktické části shrňte funkční a nefunkční požadavky na aplikaci pro hledání partnerů na české i světové štafetové závody.
4. Navrhněte wireframe obrazovek a uživatelské rozhraní reflektující výše uvedené požadavky.
5. Implementujte mobilní aplikaci pro platformu iOS, Android i pro web a popište důležité kroky implementace.
6. Srovnejte vývoj pro jednotlivé platformy z hlediska časové náročnosti případně dalších kritérií.

Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam doporučené literatury:

1. ZACCAGNINO, Carmine. Programming Flutter: Native, Cross-Platform Apps the Easy Way. 1. New York: Pragmatic Bookshelf publishing, 2020. ISBN 9781680506952.
2. LACKO, Ľuboslav.
3. Vývoj aplikací pro iOS. Brno: Computer Press, 2018. ISBN 978-80-251-4942-3.
4. THAKKAR, Mohit.
5. Google Inc. [online dokumentace]. ©2021 [cit. 30.11.2021] Dostupné z: https://docs.flutter.dev
6. Apple Inc. [online dokumentace]. ©2021 [cit. 30.11.2021] Dostupné z: https://developer.apple.com/documentation/swiftui/
7. Google Inc. [online dokumentace]. ©2021 [cit. 30.11.2021] Dostupné z: https://docs.flutter.dev/deployment/android
8. Google Inc. [online dokumentace]. ©2021 [cit. 30.11.2021] Dostupné z: https://developer.android.com/distribute/best-practices/launch/launch-checklist

Vedoucí bakalářské práce: **Ing. Radek Vala, Ph.D.**
Ústav informatiky a umělé inteligence

Datum zadání bakalářské práce: **3. prosince 2021**
Termín odevzdání bakalářské práce: **23. května 2022**

UNIVERZITA TOMÁŠE BATI VE ZLÍNĚ
Fakulta aplikované informatiky
L.S.

doc. Mgr. Milan Adámek, Ph.D. v.r.
děkan

prof. Mgr. Roman Jašek, Ph.D., DBA v.r.
ředitel ústavu

Ve Zlíně dne 24. ledna 2022

Prohlašuji, že

- beru na vědomí, že odevzdáním diplomové práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že diplomová práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové/ bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně;
- byl/a jsem seznámen/a s tím, že na moji diplomovou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – diplomovou práci nebo poskytnout licenci k jejímu využití jen připouští-li tak licenční smlouva uzavřená mezi mnou a Univerzitou Tomáše Bati ve Zlíně s tím, že vyrovnání případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;
- beru na vědomí, že pokud bylo k vypracování diplomové práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky diplomové práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem diplomové práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

že jsem na diplomové práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.

že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně, dne Filip Vabroušek, v. r.

ABSTRAKT

Tématem této bakalářské práce je mobilní aplikace Relays. Slouží ke shromáždění poptávky a nabídky členů štafetových závodů celosvětově na jednom místě. Aplikaci jsem vytvořil pro tři platformy: iOS, Android a web. Při zpracování této mobilní aplikace je použit framework SwiftUI pro platformu iOS, Flutter pro Android a React pro web. Jako technologii serverové strany je použita backendová knihovna Firebase. Aplikace Relays je dostupná na App Store a Google Play, její webová verze je zveřejněna na adrese www.relays.app.

Klíčová slova: SwiftUI, Flutter, React, Firebase, iOS, Android, web, App Store, Google Play

ABSTRACT

This bachelor thesis deals with creation of Relays mobile application. Its purpose is to gather supply and demand of relay members for relay races globally in one place. I have created the application for platforms iOS, Android and the web. This application is created using SwiftUI for platform iOS, Flutter for Android and React for web. Firebase is used as a backend library. Application Relays is available on App Store and Google Play, its web version is published at www.relays.app.

Keywords: SwiftUI, Flutter, React, Firebase, iOS, Android, web, App Store, Google Play

Děkuji Ing. Radku Valovi, Ph.D. za cenné rady, věcné připomínky a vstřícnost při konzultacích k této bakalářské práci.
Také děkuji svým rodičům za všestrannou podporu při studiu.

Prohlašuji, že odevzdaná verze bakalářské práce a verze elektronická nahraná do IS/STAG jsou totožné.

Obsah

ÚVOD

Cílem této práce bylo vytvořit aplikaci Relays pro hledání partnerů na štafetové závody. Protože jsem chtěl používání aplikace umožnit co nejširšímu počtu uživatelů, aplikaci jsem implementoval pro iOS, Android i web. U štafetových závodů bývá problémem najít vhodné partnery na daný závod. Jako příklad uvádím triatlon, skládající se z plavání, cyklistiky a běhu. Je potřeba najít vhodné sportovce, kteří ve svých disciplínách podají odpovídající optimální výkon. Po registraci do aplikace Relays má uživatel možnost vybrat si svůj závod a najít vhodné partnery. V nabídce jsou závody, které je možno jednoduše filtrovat a vyhledávat. Aplikace shromažďuje statistiku úspěšně nalezených partnerů. Většina uživatelů využívá pro komunikaci mobilní zařízení. Proto se práce zaměřuje na tvorbu mobilní aplikace Relays. Neponechává stranou ani řešení komunikace přes web. Jako ideální pro naprogramování aplikace je multiplatformní framework Flutter. To znamená, že pro operační systémy iOS, Android i web je možno použít stejný kód.

Tuto aplikaci jsem začal vyvíjet pro platformu iOS v roce 2020 s použitím knihovny SwiftUI. Po jejím zveřejnění na App Store jsem se rozhodl aplikaci nabídnout širšímu počtu uživatelů. Pro platformu Android jsem použil multiplatformní framework Flutter a její webovou verzi jsem následně naprogramoval v knihovně React, která je napsaná v jazyce JavaScript, se kterým pracuji již delší dobu.

V teoretické části práce jsem se věnoval výběru vhodné technologie a následně jsem popsal výhody a nevýhody všech tří přístupů.V praktické části práce, tedy při samotném programování, jsem kladl důraz na uživatelsky přívětivé prostředí pro klienty, kteří většinou nemají velké zkušenosti s komunikací s mobilními aplikacemi nebo webem. Popisuji zde důležité kroky implementace všech tří aplikací.

Aplikace Relays pro operační systém iOS je zveřejněna na App Store pod názvem “Relays: Find a relay partner“, pro operační systém Android je aplikaci možno najít na Google Play pod stejným názvem a její webová verze je umístěna na adrese www.relays.app.

I. TEORETICKÁ ČÁST

1 Možnosti vývoje mobilních aplikací

Mobilní aplikace jsou v současnosti velmi důležité. Podle průzkumu společnosti Appanie [9] strávili uživatelé v USA průměrně v mobilních aplikacích 2 hodiny a 15 minut. V Brazílii, Mexiku a Jižní Koreji strávili uživatelé mobilních aplikací na svém smartphonu průměrně přes 3 hodiny. 77% uživatelů uvedlo, že preferují používání mobilních aplikací před používáním počítače. Za zmínku stojí také to, že 80% uživatelů strávilo čas v aplikacích, které nejsou v horních příčkách žebříčků. Průměrný uživatel má dle studie na svém mobilním telefonu nainstalováno 35 aplikací. Hlavními obchody pro instalaci aplikací jsou App Store a Google Play [9].

Dle průzkumu společnosti Statista obsahoval Google Play 3 482 452 aplikací a App Store 2 226 823 aplikací. Díky přísné kontrole firmy Apple je ale kvalita a bezpečnost aplikací na App Store většinou znatelně vyšší. Univerzálním prostředkem pro tvorbu mobilních aplikací je deklarativní framework Flutter. Tento framework představil v roce 2018 Google jako deklarativní framework na tvorbu multiplatformních aplikací. Framework podporuje iOS, Android, macOS, Windows, Linux a Embedded zařízení. Flutter byl naprogramovaný v jazyce Dart a je open-source. Jeho zdrojový kód je k dispozici na GitHubu. [10]

Původně jsem zamýšlel Relays pro Android naprogramovat ve frameworku Jetpack Compose, ten byl ale v době začátku vývoje ve fázi alpha. K tvorbě aplikace jsem nechtěl používat stávající nativní imperativní framework, a tak jsem se rozhodl pro tvorbu aplikace ve Flutteru. Jelikož na platformě iOS využívám nativní SwiftUI a na webu React, i přes možnost využití frameworku Flutter na všech platformách jsem jej nasadil pouze na OS Android.

1.1 Nativní technologie

Nativní aplikace musí být vyvíjena pro každou platformu zvlášť. Vývojář musí vyvíjet různá řešení pro dané platformy. Mezi výhody nativních aplikací se řadí vynikající výkon, menší závislost na externích knihovnách a větší stabilita. Nativní aplikace jsou vhodné pro projekty, kde je důležitá kvalita a udržitelnost dalšího vývoje. [25]

1.1.1 Možnosti vývoje pro platformu iOS

V roce 1984 Apple uvedl jazyk Objective-C. Jazyk Objective-C si nezískal mnoho přiznivců kvůli náročné syntaxi a náchylnosti k chybám. Proto v roce 2014 Apple na konferenci WWDC14 uvedl nový programovací jazyk Swift s moderní syntaxí a automatickou prací s pamětí. Pro tvorbu aplikací Apple nejprve framework UIKit a v roce 2019 uvedl nový deklarativní framework SwiftUI. [33]

1.1.1.1 Programovací jazyk

Programovací jazyk je prostředek pro zápis lidem srozumitelného kódu, který může být počítačem převeden na posloupnost strojových instrukcí.

Swift

Jazyk eliminuje velký počet častých chyb, se kterými se programátoři potýkali v jazyce Objective-C. Jedná se například o automatickou správu paměti, kontrolu polí a číselných datových typů na přetečení. Syntaxe byla zjednodušena a umožňuje programátorovi jednodušeji vyjádřit jeho záměr. Objekty ve Swiftu nemohou mít výchozí hodnotu `nil` (nil = objekt nemá hodnotu). Pokud chceme objekt definovat jako `nil`, musí být definován jako `Optional`. V kódu pomocí otazníku indikujeme, že chování `Optional` typu rozumíme a jsme schopni jej ošetřit. Je možné využít také Playground, které nám umožňuje interaktivně experimentovat s programovacím jazykem Swift bez nutnosti kompilace projektu při každé změně. Po napsání řádku je ihned zobrazen výsledek, ať už v textové, nebo grafické podobě. Playground umožňuje vykreslovat také SwiftUI views pomocí struktury `PlaygroundPage.current.liveView`. [2]

K vývoji aplikací se používá vývojové prostředí Xcode. Toto prostředí obsahuje knihovny potřebné k vývoji aplikací a simulátor na testování aplikací během vývoje. V simulátoru je možné si zvolit typ zařízení, ať už se jedná o iPhone, iPad nebo například Apple Watch.[2]

Pokud potřebujeme zjistit, zda proměnná o typu `Optional` obsahuje hodnotu, můžeme to provést dvěma způsoby:

1) `if let r = r`

2) `guard let t = t {….}`

Pokud proměnná neobsahuje hodnotu, můžeme ve Swiftu také explicitně definovat náhradní hodnotu pomocí dvojího otazníku (`??`). Po použití této hodnoty bude uvedena namísto prázdné hodnoty.

Swift kromě standardních možností řízení toku programu (`if, else, switch`) umožňuje také vytvoření smyčky pomocí `repeat-while`. Cyklus `repeat-while` je odlišný v tom, že se nejprve provede operace uvnitř bloku `repeat` a až poté se zkontroluje, zda podmínka uvnitř bloku `while` platí. Proběhne tedy vždy minimálně jedenkrát. Protokoly slouží jako předpis pro dané třídy, obsahují však pouze jen definice atributů a metod, které nejsou implementovány. Jedná se o šablonu, která předepisuje určité funkce a proměnné. Jedním z nejdůležitějších protokolů ve SwiftUI je protokol `View`, který obsahuje proměnnou `body` pro tvorbu našeho uživatelského rozhraní. Swift podporuje dědičnost ve třídách pomocí zápisu za dvojtečku. Při programování se nejprve definuje rodičovská tříd `Person` s konstruktorem a property jménem. Od této třídy se vytvoří odvozená třída `User:Person`, která bude mít navíc oproti uživateli heslo. Uvnitř konstruktoru `init` ve druhé třídě se zapíší opakující se proměnné do volání funkce `super` a nové proměnné jsou přiřazeny pomocí `self`. Následně je vytvořena této třídy. Argumenty jsou přeneseny do konstruktoru jako volání funkce. [5]

```
class Person {
var name: String
    init(name: String){
        self.name = name
```

```
        }
    }

    class User: Person {
    var password: String
    init(username: String, password: String){
        self.password = password
        super.init(name: username)
    }
    }

    let me = User(username: “Jméno”, password: “Heslo”);
```

1.1.1.2 Knihovny

Knihovny jsou předprogramované bloky kódu, které ulehčují a zrychlují vývoj aplikací. V případě vývoje mobilních aplikací se jedná se například o tlačítka, seznamy, menu pro výběr dat, navigační rozhraní. Při tvorbě nativní aplikace na iOS jsou k dispozici hlavně dvě možnosti: UIKit a SwiftUI [34].

UIKit

Nejprve Apple uvedl framework Foundation obsahující nástroje pro práci s daty, textem a dalšími běžnými datovými typy a framework UIKit pro tvorbu uživatelských rozhraní. Oba frameworky jsou naprogramovány v jazycích Objective-C a Swift. Framework UIKit je možné používat jak s jazykem Swift, tak s jazykem Objective-C. Knihovna UIKit byla představena v roce 2008, krátce po uvedení prvního telefonu firmy Apple - iPhone. Framework využívá imperativní způsob tvorby uživatelského rozhraní. UIKit obsahuje scény pro tvorbu obrazovek a elementy pro reprezentaci jejich obsahu. Každá obrazovka má svoji vlastní instanci třídy `UIViewController`, který řídí danou obrazovku. Základním stavebním kamenem uživatelského rozhraní je třída `UIView`, ze které ostatní uživatelské prvky dědí. Veškeré kreslení na obrazovku probíhá pomocí metody `draw(in: Rect)`. Pro překreslení uživatelského rozhraní po změněně stavu aplikace je nutné zavolat speciální funkce. [2][6]

Uživatelské rozhraní je možno tvořit dvojím způsobem:
1) Graficky pomocí Storyboardu a následné propojení s kódem.
2) V kódu, kdy lze vytvořit všechny prvky uživatelského rozhraní v jazyce Swift nebo Objective-C.

SwiftUI

SwiftUI je nový framework pro tvorbu uživatelských rozhraní představený na vývojářské konferenci WWDC19. Na rozdíl od frameworku UIKit je naprogramován v jazyce Swift. Využívá nový, deklarativní zápis kódu a zvyšuje jeho přehlednost. Prvky uživatelského rozhraní nejsou implementovány jako třídy pomocí datového typu `class`, ale jako struktury pomocí datového typu `struct`. [4]

Každý View ve SwiftUI může obsahovat proměnnou definovanou pomocí Property Wrapperu `@State`. Po změně hodnoty této proměnné se automaticky ta část uživatelského rozhraní, kde je tato proměnná použita, překreslí. Tuto proměnnou můžeme navázat do jiných views pomocí property `@Binding`. Pokud se tato proměnná změní, změní se i hodnota ve `View`, do které byla tato hodnota navázána. SwiftUI neobsahuje všechny prvky ze staršího frameworku UIKit, a tak je možné je implementovat pomocí protokolů `UIViewRepresentable` a `UIViewControllerRepresentable`. V klasické dědičnosti je využíván vztah „IS", což v překladu znamená „JE". Od rodičovské třídy `UIView` jsou následně odvozeny všechny ostatní prvky, jako například `UIButton`, `UIViewController` nebo `UIPickerView`. [34]

Na rozdíl od staršího frameworku UIKit `View` není třída, ale protokol. Protokol vyjadřuje vztah „CAN DO" což česky znamená „UMÍ". V jazyce Swift je protokol předpis pro danou třídu. Předepisuje jaké metody nebo proměnné tato třída obsahuje. Framework využívá principu Composition over Inheritenance, neboli kompozice namísto dědičnosti. Ve SwiftUI je využívaný protokol `View`, který obsahuje povinnou property `body`. Nejedná se přímo o objekt jako ve frameworku UIKit, ale pouze o jeho předpis. O samotné vykreslování se stará platforma, na které aplikace běží. Vývojáři SwiftUI nevytvořili všechny komponenty na zelené louce, ale velké část komponentů interně používá komponenty z knihovny UIKit. [34]

Například element `List` používá interně element `UITableViewController` z frameworku UIKit. Pro reprezentaci reagování na změny stavů v aplikaci SwiftUI používá interně framework AttributeGraph. SwiftUI na základě změn dat v této reprezentaci efektivně překreslí daná Views. Tento framework byl naprogramován v jazyce C++. SwiftUI podporuje všechny operační systémy firmy Apple - iOS, macOS, watchOS, tvOS. Výhodou tohoto frameworku je že se automaticky přizpůsobí každé platformě. Jako příklad mohu uvést vzhled elementu `Picker`. `Picker` slouží k výběru prvků se seznamu. Na platformě iOS element `Picker` vypadá jako kotoučová roleta a na platformě macOS jako rozklikávací menu přizpůsobené na klik myši. [35]

1.1.2 Možnosti vývoje pro platformu Android

Android je operační systém pro mobilní zařízení. Tento systém byl původně vytvořený firmou Android Inc. pro digitální fotoaparáty. V roce 2004 se projekt stal operačním systémem pro mobilní

telefony. Tato společnost byla v roce 2006 zakoupena společností Google. [26]

1.1.2.1 Jazyky

Původním jazykem pro vývoj na platformě Android byla Java. Tento jazyk nepřinášel vývojářům dostatečný komfort kvůli nízké efektivitě při psaní kódu, a proto přešel Google na pokročilejší jazyky.[36]

Java

Java je objektově orientovaný programovací jazyk, který byl představen v roce 1995. Je nezávislý na dané platformě díky kompilaci do byte kódu. Jeho kompilátor produkuje architektonicky nezávislý objektový soubor, který umožňuje spuštění kódu na různých procesorech. Zaměřuje se na odhalení chyb během kompilace. [37]

Kotlin

Programovací jazyk Kotlin byl vytvořen v roce 2011 vývojáři z firmy JetBrains. Tento jazyk umožňuje zápis kódu pomocí moderní syntaxe. Ekvivalentní kód zabere přibližně o 40% méně řádků kódu než v kód v jazyce Java. V roce 2019 v něm Google vytvořil nový framework JetPack Compose. [13]

Dart

Programovací jazyk Dart je používaný pro vývoj ve frameworku Flutter, ve kterém byla vyvinuta aplikace Relays. Dart je stejně jako Swift type-safe, což znamená že hodnota proměnné vždy musí odpovídat jejímu statickému typu. Dart také podporuje sound null safety, což znamená že hodnota nemůže nabýt hodnoty `null` neočekávaně. Dart podporuje dědičnost pomocí slova `extend`. [15]

1.1.2.2 Knihovny

Aplikace pro Android se původně vyíjely v nativních nástrojích na vývoj softwaru - Software Development Kit (SDK). V průběhu vývoje se ukázaly problémy se strukturou původní kódu, a proto byl vyvinut nový framework JetPack Compose. [27]

Nativní SDK

Rok po představení systému Android byla vývojářům v roce 2009 uvolněna knihovna Android SDK. K tvorbě aplikací se používal jazyk Java nebo Kotlin a pro tvorbu uživatelského rozhraní jazyk XML. [27]

JetPack Compose

Android původně používal starší UI framework pro tvorbu aplikací, ale v roce 2019 byla vytvořena nová knihovna - Jetpack Compose. Knihovna podobně jako SwiftUI a Flutter využívá deklarativní a expresivní zápis uživatelského rozhraní. [14]

1.1.3 Možnosti vývoje pro webovou platformu

Do frontend patří technologie, které fungují na straně webového prohlížeče. S touto vrstvou přímo komunikuje uživatel webové aplikace.

Do backendu patří technologie, se kterými uživatel aplikace nepřichází přímo do styku a nevidí je. Exekuce kódu této vrstvy se provádí na serverech. Tato vrstva zásobuje frontend daty. [28]

1.1.3.1 Backendové technologie - jazyky

SQL

SQL (structured query language) je strukturovaný dotazovací jazyk, který je často používaný pro práci s relačními databázemi pomocí příkazů podobných anglickému jazyku. Pro vytváření databáze se používá příkaz CREATE DATABASE Users. Nevýhodou této technologie je nízká škálovatelnost. [16]

PHP

PHP (PHP: Hypertext preprocesor) je jazyk serverové strany, který poslední dobou upadá v popularitě. Tento jazyk umožňuje komunikovat s SQL databází a dynamicky vykreslovat uživatelské rozhraní pomocí HTML na straně klienta [17][29]

1.1.3.2 Backendové technologie - knihovny

Node.js

Node.js je webové rozhraní, které umožňuje běh javascriptového kódu na serveru (backend). Tato technologie je během vývoje používána například při doručování notifikací na iOS a Android aplikace pomocí služby Firebase Cloud Functions. Příkazy pro zasílání notifikací se píší do textového souboru a pomocí příkazu `firebase deploy --only functions` se nahrají na server. [18]

Firebase

Firebase se vyvinula z Envolve, předchozího startupu, který byl založen za účelem integrování chatovací funkcionality do webových stránek. Uživatelé knihovnu využívali k jiným účelům než k chatování - například k synchronizování aplikačních dat jako stav her.

V roce 2011 vznikla Firebase jako nezávislá společnost a vývojářům byla uvolněna v roce 2012. Tato technologie využívá key-value store a na rozdíl od SQL nemusí dodržovat pevnou stukturu dat. Webová verze knihovny Firebase využívá jazyk JavaScript. Do této databáze lze vložit jakoukoli hodnotu pod libovolným klíčem. [12]

1.3.1.3 Frontendové technologie - jazyky

JavaScript

JavaScript je programovací jazyk, který vytvořil Brendan Eich z tehdejší společnosti NetScape v květnu 1995. Mezi jeho výhody se řadí všeobecná podpora v prohlížečích a možnost tvorby jak frontendových, tak i backendových systémů. Jeho hlavní nevýhodou je slabá typová kontrola. JavaScript podporuje dědičnost pomocí slova `extend`. Podobně jako ve Swiftu se zděděné argumenty přenášejí ve funkci `super`. Pro vytvoření instancí tříd se na rozdíl od Swiftu musí použít klíčové slovo `new`. Jedná se o syntactic-sugar nad funkcionálními prototypy objektů používanými v standardu ECMAScript 5. [24]

1.3.1.4 Frontendové technologie - knihovny

Angular

Angular je framework od společnosti Google primárně využívající jazyk TypeScript se silným typováním. Je založen na architektonickém principu MVC (Model-View-Controller) a byl uveden v září 2016. Framework obsahuje komponenty pro tvorbu škálovatelných webových aplikací a kolekci integrovaných knihoven. [19]

Vue

Vue je knihovna vyvinutá Evanem You. Jako architektonický vzor je používaný model MVVM (Model-View-ViewModel). Knihovna podporuje tvorbu logiky aplikace pomocí direktiv uvnitř HTML tagů jako například v-if pro podmíněné skrytí elementu. [20]

ASP-NET

ASP-NET je framework na tvorbu webových aplikací vyvinutý společností Microsoft. Využívá zápis HTML a C# kódu do jednoho souboru pomocí technologie Blazor. Je schopna úzké spolupráce s frameworkem Entity Framework Core pro mapování C# tříd na databázové tabulky. [21]

React

React je javascriptová knihovna pro tvorbu frontendu webových aplikací. Tuto knihovnu vyvinul Jordan Walke ze společnosti Facebook. Knihovna byla uvolněna veřejnosti v květnu 2013. Stav v Reactu se tvoří pomocí objektu state. Funkce `setState` nám umožní změnit stav aplikace a překreslit danou část uživatelského rozhraní. React vytváří komponenty pomocí volání funkce React.createElement. Jelikož je zápis kódu tímto způsobem poměrně složitý a nepřehledný, React využívá syntaxi JSX pro zápis HTML v JavaScriptu. JSX je tzv. syntactic sugar (zjednodušený zápis) prvků uživatelského rozhraní. Tagy JSX jsou poté pomocí kompilátoru Babel zkompilovány do klasického JavaScriptu. Kompilaci je možné zajistit dvojím způsobem:
1) překompilováním kódu do standardu ES5 pomocí příkazového řádky a nahrání zdrojových souborů
2) překompilováním souborů v prohlížeči pomocí knihovny na straně uživatele.

K tvorbě komponentu v Reactu můžeme využít dva způsoby. Prvním z nich je rozšíření třídy `Component` nebo použití funkce. Zde rozšíříme třídu `React.Component` a definujeme v ní konstruktor. Uvnitř konstruktoru vytvoříme objekt `state`, ve kterém bude ukládán stav komponentu. Dále vytvoříme metodu `render`, která bude vracet uživatelské rozhraní našeho komponentu pomocí zápisu JSX.

Druhým způsobem je zápis pomocí funkce hook. Definujeme funkci `Counter` a uvnitř ní definujeme funkci `useState`, která nám pomocí dělení proměnných dá k dispozici proměnnou `count` a funkci `setCount` pro aktualizaci hodnoty této proměnné. Proměnná `count` nám bude vracet danou hodnotu počítadla a voláním funkce `setCount` se bude tato hodnota zvyšovat. Uvnitř tlačítka `<button>` si tuto hodnotu vypíšeme. [22]

```
function Counter(){
    const [count, setCount] = React.useState(0);
    return (
        <div>
            <p>You clicked {count} times</p>
            <button onClick={() => setCount(count + 1)}>
                Click me
            </button>
        </div>
```

Obrázek 1: React hook

1.2 Multiplatformní technologie

Multiplatformní vývoj umožňuje tvorbu jedné aplikace, která bude schopna běhu na více operačních systémech. Pozitivní vlastnostní vlastností multiplatformního vývoje je zachování jednotného designu a také úspora času. U složitějších aplikací je často potřeba využít i nativní kód.

1.2.1 Ionic

Ionic je open-source framework pro tvorbu progresivních mobilních aplikací pomocí HTML5, CSS a JavaScriptu. Umožňuje volání nativních funkcí jednotlivých platforem pomocí JS bridge a integraci s populárními JS frameworky jako React, Angular, nebo Vue. Framework je ale méně výkonný než Flutter. [30]

1.2.2 React Native

React Native je open-source knihovna pro tvorbu aplikací pro Android a iOS. Jako základ slouží webová knihovna React. React Native obsahuje bloky jako `<Text>`, `<Image>` a `<View>`, které jsou mapovány na jednotlivé nativní komponenty na dané platformě. Tento framework využívá externích knihoven, které slouží například k tvorbě navigace. [31]

1.2.3 Flutter

V roce 2018 Google představil deklarativní framework na tvorbu multiplatformních aplikací - Flutter. Framework podporuje iOS, Android, macOS, web, Windows, Linux a embedded zařízení. Flutter byl naprogramovaný v jazyce Dart a je open-source. Jeho zdrojový kód je k dispozici na GitHubu. [3]

Flutter funguje jako vrstvy nezávislých knihoven. Každá vrstva knihovny je navržena tak, aby byla nezávislá na té předchozí. Knihovna Material kreslí prvky uživatelského rozhraní, které vypadají jako nativní Android prvky a knihovna Cupertino kreslí prvky, které vypadají jako nativní iOS aplikace. Další podstatnou součástí frameworku Flutter je Engine SKIA, který zajišťuje vykreslení daného uživatelského rozhraní. Tento framework byl převážně naprogramován v jazyce C++. Uživatelské rozhraní je ve Flutteru tvořeno po jednotlivých komponentech pomocí widgetů. [23]

Knihovna `dart:ui` obaluje interní C++ kód komponent ve třídách jazyka Dart, se kterým pracuje programátor. Poslední součástí je Embedder, jehož úkolem je vložení aplikace do vykreslovacího prostředí na dané platformě. Framework obsahuje sadu widgetů pro tvorbu aplikací, které můžeme rozdělit do dvou typů: Stateless a Stateful. Stateless widget se překreslí při startu aplikace a při překreslení rodičovského widgetu. Stateful widget se překreslí po zavolání funkce `setState`. Uvnitř funkce `setState` dosadíme název proměnné a hodnotu, na kterou se tato proměnná má změnit. Flutter poté překreslí potřebné widgety tak, aby data v nich odpovídala novým hodnotám proměnných. [1]

Hot reload slouží ke aktualizaci stavu aplikace, bez nutnosti její rekompilace. Je například možné změnit pozadí a klávesovou zkratkou CTRL+ S aplikaci obnovit. Další výhodou této metody

je zachování stavu mezi aktualizacemi uživatelského rozhraní. [23]

Před začátkem vývoje je nutné nainstalovat Flutter SDK. Používám macOS, a proto popíšu instalační instrukce právě pro tuto platformu. Nejprve je potřeba stáhnout zip archiv s frameworkem z oficiálního webu Flutteru. Archiv jsem rozbalil ve svém domovském adresáři a spustil jsem následující příkaz: `nano ~/.bash_profile.`

Následně jsem vložil příkaz `export PATH=$PATH:/example/path/to/flutter/bin`. Tuto cestu jsem nahradil svojí vlastní cestou. Poté jsem soubor uzavřel pomocí klávesové zkratky ctrl + X a uložil jej pomocí stisku klávesy Y. Abych mohl příkaz `flutter` používat, potřeboval jsem provést refresh konfigurace terminálu pomocí příkazu `source ~/.bash_profile`.

Pro vývoj jsem použil Android Studio, do kterého jsem si nainstaloval plug-in Flutter. Poté jsem v Android Studiu vytvořil nový Flutter projekt a spustil jsem ho pomocí příkazu `flutter run`. Během vývoje jsem převážně využíval reálná zařízení, protože mi lépe umožňují přiblížit se prožitku uživatele aplikace. Framework obsahuje několik příkazů, pomocí kterých můžeme komunikovat s frameworkem. Například: `flutter doctor` zkontroluje, jestli je potřeba doinstalovat některé závislosti. Framework se upgraduje příkazem `flutter upgrade`.

Gradle je ve Flutteru využíván pro automatizaci a sestavování programu. Gradle byl naprogramován v jazyce C++ a sám využívá Java Virtual Machine. Úlohy pro sestavení aplikace lze spustit sériově nebo paralelně. Gradle je open source a jeho kód je dostupný na GitHubu. Jeho první verze vyšla v březnu 2008. [23]

1.3 Asynchronní programování

V některých případech je nutné počkat na výsledek asynchronní operace, nebo naopak paralelně pokračovat v exekuci dalšího kódu. Výhodou asynchronních funkcí, které jsou používány v mobilních a síťových aplikacích je, že tyto funkce jsou schopné se pozastavit během provádění a následně pokračovat. [32]

1.3.1 Swift

Ve Swiftu se pro asynchronní může používat klíčové slovo `await`, nebo `callback`. Ve své aplikaci používám právě druhou variantu. Callback se ve Swiftu vytvoří například pomocí zápisu `@escaping`. Toto klíčové slovo indikuje to, že funkce může vrátit hodnotu i poté, co proběhne. Zápis je ve formátu `func benchmark(_ closure: () -> Void)`. [5]

1.3.2 Dart

Ve Flutteru se o asynchronní programování stará knihovna dart:async. Pokud chceme funkci označit jako asynchronní, jako datový typ funkce použijeme `Future<void>`, napíšeme název funkce včetně kulatých závorek a na konec připíšeme klíčové slovo `async`. Pokud tuto funkci zavoláme, další kód nebude čekat na doběhnutí této funkce, ale poběží paralelně. V případě, že budeme chtít počkat, něž asynchronní funkce doběhne, použijeme před názvem asynchronní funkce klíčové slovo `await`. Funkci, uvnitř které používáme `await`, musíme taktéž označit pomocí klíčových slov

`await` a `async`. [1][23]

1.3.3 JavaScript

Na webu se často k asynchronnímu programování používá asynchronní objekt Promise. Promise je objekt reprezentující selhání nebo dokončení asynchronní operace a její návratovou hodnotu. Jedná se o zástupce hodnoty, jejíž hodnota není známa v době tvorby Promise. Může nabýt jednoho ze tří stavů:

`pending` - výchozí stav promise nedokončena ani neodmítnuta
`fulfilled` - operace byla úspěšně dokončena
`rejected` - označuje selhání operace

Promise může nabýt stavů `fulfilled` nebo `rejected`. Tento objekt vrací dvě funkce: `resolve` a `reject`. Pokud chceme vrátit hodnotu, použijeme `resolve`, pokud chceme indikovat že při běhu došlo k chybě, použijeme `reject`. Následně můžeme funkci zavolat a na její konec zřetězit handlery `then`. Po zavolání funkce `resolve` se spustí právě handler `then`. Na konci řetězce můžeme použít handler `catch`, který je schopný zachytit veškeré chyby, které mohou vzniknout při běhu programu. [11]

2 Výběr technologií

Aplikaci Relays by bylo možné pro všechny tři platformy (iOS, Android a web) vytvořit pouze v multiplatformním frameworku Flutter. Původním záměrem bylo vytvoření aplikace pouze pro iOS a až po jejím zveřejnění na App Store jsem se rozhodl ji rozšířit pro platformy Android a web. Pro Android jsem zvolil Flutter, protože jsem nechtěl k vývoji používat stávající imperativní SDK a Jetpack Compose byl k dispozici pouze v alpha verzi. Pro React ve webové verzi jsem se rozhodl kvůli standardnímu přístupu v Javascriptu, se kterým pracuji již delší dobu.

2.1 Výhody SwiftUI

Celou strukturu aplikace můžeme jednoduše číst, upravovat a udržovat. Syntaxe je expresivní, čistá a čitelná, což přispívá k rychlé orientaci vývojáře v kódu. Vývojář může pomocí canvas v Xcode editovat vizuálně aplikaci bez nutnosti rekompilace. Swift má strukturovaný silný typový systém, který umožňuje snadné odstranění chyb. Swift umožňuje programátorovi vyhnout se celým třídám chyb, jako jsou neinicializované proměnné, přetečení proměnných a nedovolené formátování textových řetězců.

2.2 Nevýhody SwiftUI

SwiftUI v současnosti neobsahuje všechny systémové komponenty z frameworku UIKit. Implementace těchto chybějících komponent může být v případě některých případech časově náročná. Použití tohoto frameworku je možné až od verze iOS 13 a výše.

2.3 Výhody Flutteru

Flutter umožňuje zrychlit čas vývoje aplikace používáním jednoho kódu, který běží na všech platformách. Flutter poskytuje bohatou knihovnu předdefinovaných widgetů, a proto je vývoj aplikací velmi rychlý. Obsahuje dvě knihovny widgetů: Material design pro Android aplikace a Cupertino (iOS) pro iOS aplikace. Také umožňuje vytváření vlastních widgetů. Renderovací engine Flutter - Skia je velmi výkonný a efektivní a umožňuje vytvářet pokročilé efekty a animace. Dart nepoužívá bridge jako většina multiplatformních JS frameworků, ale kompiluje se přímo do nativního kódu. Toto má za následek rychlé spuštění aplikace a možnost tvorby animací se zobrazovací frekvencí až 120 fps.

Aplikace vytvořené ve Flutteru podporují Accessibility, tedy možnost tvořit aplikace pro znevýhodněné uživatele. Flutter SDK podporuje různé měny a regionální požadavky. Flutter využívá hot reload, který umožňuje provádět změny v aplikaci za běhu. Změny v aplikaci se promítnou během několika vteřin. Toto přispívá k rychlému řešení chyb objevených během vývoje. Hot reload tak umožňuje vývojáři rychleji iterovat na designu aplikace podle požadavků grafického návrháře.

2.4 Nevýhody Flutteru

Největší nevýhodou je potřeba vysokého množství závislostí a build systém Gradle. Pokud se změní verze Gradle nebo se aktualizuje knihovna, která závisí na jiné než na nainstalované verzi, může toto vést k chybám při kompilaci a archivování pro Google Play/ App Store. Aplikace vytvořené ve Flutteru jsou často větší než jejich nativní protějšky. Toto je z velké části zapříčiněno tím, že používají předdefinované widgety namísto nativních. Nejnovější vylepšení v systémech iOS a Android budou nejprve dostupná v jejich nativních knihovnách a až poté ve Flutteru.

2.5 Výhody Reactu

React využívá virtuální reprezentaci uživatelského rozhraní (virtuální DOM). Díky němu mohou vývojáři tvořit rychlé aplikace, které pružně reagují na změny v databázi a uživatelské akce. V Reactu se dají tvořit znovupoužitelné komponenty, které se dají využít i v budoucích projektech. Vývoj aplikace se tím zjednoduší a zrychlí.

React je open-source (má otevřený zdrojový kód) a má okolo sebe rozsáhlou komunitu vývojářů. Hledání informací o této knihovně je často snadné a efektivní. Aplikace vytvořené v Reactu jsou velmi dobře indexovány Googlem a mají vysoké SEO ohodnocení (ohodnocení v prohlížečích). React je také jednodušší na naučení než velká část konkurenčních frameworků. React využívá one-way data flow, což označuje jednosměrný tok dat a umožňuje snazší pochopení funkcionality aplikace.

Komponent v Reactu se tvoří kompletně včetně chování a struktury v javascriptovém souboru a je v něm možné vytvořit i CSS styly. Není potřeba přepínat mezi HTML a CSS soubory, což přispívá k větší produktivitě během vývoje.

2.6 Nevýhody Reactu

React se velmi rychle vyvijí a proto dosud nevznikla rozsáhlá a podrobná dokumentace knihovny. Často může být problém se orientovat v rychlém tempu vydávání nových verzí. React používá v zápisu kódu formát JSX, který umožňuje obalit volání `document.createElement` do tagů HTML. Pro začátečníky toto může být nepřehledné.

II. PRAKTICKÁ ČÁST

3 Úvod do aplikace z hlediska uživatele

Jak již bylo řečeno v úvodu, tato aplikace vyhledává partnery na štafetové závody. U těchto závodů bývá problémem najít vhodné partnery na daný závod. Jedná se například o specifické dovednosti (jízda na kajaku, paragliding, bruslení). Aplikace zohledňuje výkonnost sportovce. Při využití aplikace Relays má uživatel možnost vybrat si závod a najít pro tento závod vhodné partnery. Aplikace obsahuje seznam závodů s uvedenými disciplínami, které je možno jednoduše filtrovat a vyhledávat.

3.1 Požadavky na aplikaci

Před programováním aplikace je nutno definovat funkční a nefunkční požadavky. Funkční požadavky zajišťují plynulou a bezchybnou funkci aplikace. Nefunkční požadavky určují požadavky na bezpečnost a kompatibilitu s příslušnými verzemi operačních systémů na mobilních zařízeních a ve webových prohlížečích. Obě skupiny požadavků jsem rozdělil do následujících tabulek.

3.1.1 Funkční požadavky

ID požadavku	Popis požadavku
FUN-001	Aplikace musí být schopna zaregistrovat uživatele.
FUN-002	Aplikace musí být schopna přihlásit uživatele..
FUN-003	Aplikace musí být schopna odhlásit uživatele.
FUN-004	Aplikace musí být schopna uživatele přihlásit na závod.
FUN-005	Aplikace musí naběhnout do 4 sekund.
FUN-006	Aplikace musí být schopna odebrat uživatele ze závodu.
FUN-007	Aplikace se musí uživatele zeptat na důvod odhlášení ze závodu.
FUN-008	Aplikace musí umožnit filtrovat závody.
FUN-009	Aplikace musí umožnit vybrat a poslat obrázek v chatu.
FUN-010	Aplikace musí být schopna zobrazit doručené zprávy.
FUN-011	Aplikace musí být schopna změnit data uživatele

Tabulka č.1: Funkční požadavky

3.1.2 Nefunkční požadavky

ID požadavku	Popis požadavku
NFUN-001	Popisují chování aplikace, které nemají tak velký vliv na aplikaci.
NFUN-002	Aplikaci musí být kompatibilní s verzí iOS n-1. Toto znamená, že aplikace bude dostupná na aktuální verzi iOS a na té předchozí.
NFUN-003	Kompatibilita s Android verze 5 a vyšší. Důvodem je menší počet uživatelů na novějších verzích operačních systémů než u systému iOS.
NFUN-004	Ve webové verzi je důležitá podpora všech hlavních webových prohlížečů (Chrome, Safari, Firefox) a zabezpečení SSL certifikátem.
NFUN-005	Aktuální verze knihoven React a Firebase.

Tabulka č.2: Nefunkční požadavky

3.2 Implementace aplikace

Přestože je možno aplikaci naprogramovat pro všechny tři operační systémy v multiplatformním frameworku Flutter, rozhodl jsem se tento framework použít pouze pro platformu Android. Pro operační systém iOS jsem využil framework SwiftUI a pro webovou aplikaci knihovnu React.

3.3 Interakce uživatele s aplikací

Přihlašovací obrazovka

Při prvním spuštění aplikace se uživateli zobrazí seznam závodů. Tento seznam si může uživatel jen prohlédnout. Pokud má zájem o přihlášení na závod, je nutné se zaregistrovat. Roletu se závody skryje pomocí tlačítka „Close“. Tím se dostane na přihlašovací obrazovku s textovými poli pro zadání e-mailu a hesla. Po stisku tlačítka „Log-in“ aplikace přihlásí uživatele.

Registrační obrazovka

Uživatel se může zaregistrovat stiskem tlačítka „Create an account“. Zde může uživatel také nepovinně zvolit profilový obrázek. Po stisku tohoto tlačítka je uživatel zaregistrován a přesměrován na úvodní stránku aplikace. Registrační obrazovku tvoří tři textová pole:

1) E-mail
2) Uživatelské jméno (Nickname)
3) Heslo (Password)

Obrazovka s vybranými závody

Na této obrazovce je seznam závodů, které si uživatel vybral. Při prvním vstupu do aplikace je tato obrazovka prázdná, a obsahuje textovou instrukci, která říká, že pro výběr závodu je potřeba stisknout tlačítko „+ Race“ v pravém horním rohu obrazovky.

Uživatel se může ze závodu odhlásit z každé kategorie zvlášť nebo najednou ze všech disciplín daného závodu přejetím prstu po obrazovce se závody. Po přejetí prstu po obrazovce se aplikace uživatele zeptá na důvod odhlášení ze závodu.

Při výběru závodu jsou uživateli k dispozici dvě možnosti - může se nabízet jako sportovec anebo hledat dalšího sportovce. Pokud dalšího sportovce do štafety pouze hledá, nemusí zadávat čas nebo potvrzovat, že je schopen se účastnit některé z disciplín. Pokud se chce ale do štafety nabídnout k dalšímu sportovci, musí zadat čas a nebo potvrdit schopnost účastnit se některé disciplíny.

Obrazovka s výběrem disciplín

Po výběru závodu tato obrazovka obsahuje vybrané závody uživatele. Po stisku jednotlivé buňky seznamu je uživatel přesměrován na obrazovku, kde si může zvolit, zda si chce zobrazit uživatele, kteří hledají partnera na disciplínu nebo uživatele, kteří chtějí danou diciplínu absolvovat.

Obrazovka se seznamem uživatelů

Po stisku tlačítka s touto disciplínou se zobrazí seznam uživatelů. S kterýmkoli z uživatelů lze zahájit chat stisknutím dané buňky seznamu.

Obrazovka s chatem

Aplikace umožňuje standardní komunikaci pomocí chatu. Kromě textových zpráv je možné posílat i obrázky z galerie uživatele. Obrázky jsou před nahráním zmenšeny, aby nezabíraly příliš mnoho úložného prostoru. Po výběru obrázku se vytvoří uživatelské rozhraní pro výběr obrázku a tlačítko pro jeho odeslání.

Obrazovka se seznamem všech uživatelů

Aplikace obsahuje obrazovku se seznamem všech uživatelů - Competitors. Jsou zde zobrazení všichni zaregistrování uživatelé aplikace bez ohledu na vybraný závod nebo zvolené disciplíny. Po stisku buňky s uživateli lze opět zahájit chat. Uživatelé se dají filtrovat dle disciplín.

Obrazovka Messages

Na obrazovce „ Messages" se zobrazí všechny zprávy od ostatních uživatelů. Po stisku dané zprávy lze s tímto uživatelem zahájit chat.

Obrazovka Settings

Tato obrazovka umožňuje uživateli změnu časů, potvrzení schopnosti ovládat komplexní sporty a změnu profilového obrázku.

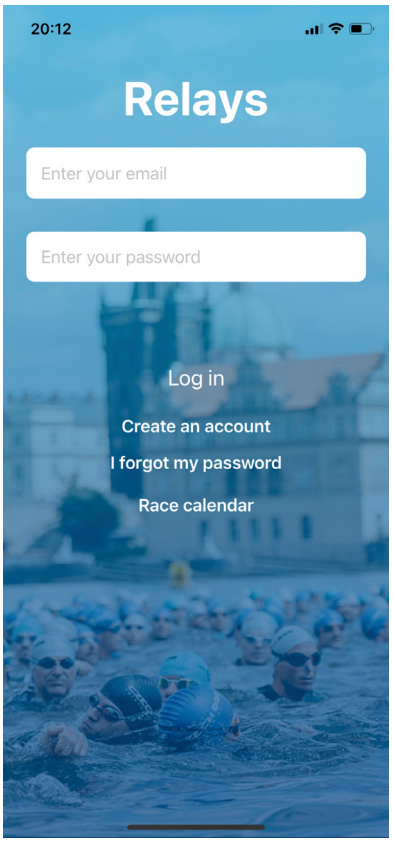
3.3.1 Implementace pro operační systém iOS

Jako vstupní bod do aplikace slouží soubor SceneDelegate. Vytvořil jsem View `MultiDecider` s `ObservableObject`, který obsahuje property s identifikátorem právě načtené obrazovky. Pomocí hodnoty ObservableObjektu se aplikace rozhodne, v jakém view se právě nachází a přepne do něj uživatele na základě hodnoty v `UserDefaults`, ve které je zapsána informace o tom, zda je uživatel na tomto zařízení registrován. Jelikož SwiftUI nepodporuje WebView, musíme ho do aplikace přidat pomocí `UIViewRepresentable`. Protokol `UIViewRepresentable` vyžaduje dvě metody. Tou první je `makeUIView(context: Context)`, který bude vracet instanci třídy `WKWebView`. Tou druhou je `updateUIView(_ uiView: WKWebView, context: Context)`, která bude pomocí `uiView.load(request)` načítat request do webového prohlížeče.Této funkcionality je využito na chatovací obrazovce po rozkliknutí zprávy obsahující danou URL. Dále už jen stačí napsat příkaz `pod install` a otevřít vygenerovaný `.xcworskapce` projekt. Do projektu přidáme inicializační kód, který ním webová konzole Firebase vygeneruje. Po spuštění tohoto projektu už si jen stačí v konzoli ověřit, že aplikace komunikuje se servery Firebase.

3.3.1.1 Přihlašovací obrazovka

Nejprve si ověřím, jestli je uživatel zaregistrovaný pomocí metody `UserDefaults` a pokud ne, přepnu ho na přihlašovací obrazovku. Přihlašovací obrazovka se sestává ze dvou textových polí

pro heslo a e-mailovou adresu. Na obrazovce se nachází i přihlašovací tlačítko. Pro uživatelské jméno i heslo definujeme proměnné typu `String` o typu property wrapper `@State`. Tyto proměnné dále navážeme do textových polí. Textové pole vytvoříme pomocí view `TextField(text: $someText)` za argument `text` dosadíme naši proměnnou. Po změně hodnoty v tomto textovém poli bude změněna i naše hodnota v proměnné.Uživatele přihlásíme pomocí funkce `Auth.auth().signIn(withEmail: email, password: password)`.



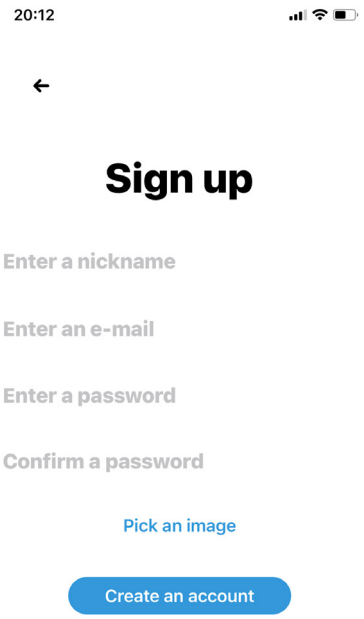
Obrázek 2: Přihlašovací obrazovka

3.3.1.2 Registrační obrazovka

Aplikace naprogramovaná ve Firebase se ovládá ve webové konzoli Firebase na adrese console. firebase.google.com. V konzoli jsem stisknul na tlačítko iOS. Vytvořil jsem nový projekt a zadal jsem název projektu „ Relays". Dále jsem do konzole zadal BundleID aplikace. Vygeneroval jsem soubor info.plist a přetáhl jsem jej do kořenové složky projektu. Potom jsem na svém MacBooku s macOS otevřel terminál a přesunul se do složky, která obsahovala můj projekt. Příkazem `pod init` jsem vygeneroval soubor, do kterého se zapíší názvy knihoven: Dále už jen stačí provést příkaz `pod install` a otevřít vygenerovaný .xcworskapce projekt. Do projektu přidáme inicializační kód, který nám konzole vygeneruje. Po spuštění tohoto projektu už si jen stačí v konzoli ověřit, že aplikace komunikuje se servery Firebase. Po stisku tlačítka SignUp se zavolá funkce, `Auth.auth().createUser(withEmail: _, password: _)`, která vytvoří na serveru nový uživatelský účet.

Pro tvorbu této obrazovky jsem využil view `TextField`, na který jsem pomocí Binding navázal příslušné proměnné. Dále jsem pomocí třídy odpovídající protokolu

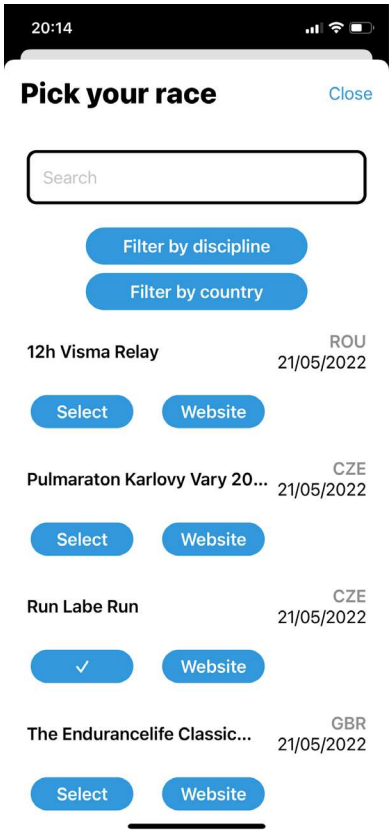
`UIViewRepresentable` vytvořil funkci na výběr obrázků. Pomocí třídy `Resizer` uživatelský obrázek zmenším na velikost 400 × 400 obrazových bodů, aby nezabíral hodně úložného prostoru. Po registraci je uživatel přesměrován na úvodní stránku aplikace.



Obrázek 3: Registrační obrazovka

3.3.1.3 Obrazovka pro výběr závodu

K tvorbě obrazovky závodů jsem vytvořil View s elementem List, který uživateli zobrazí seznam závodů. Tento element je naplněný z databáze závodů na webu Firebase. Entita databáze, kde jsou závody uloženy, se nazývá „ Races“. Každý závod má v databázi klíč `users`, který obsahuje identifikátory uživatelů zaregistrovaných na závod. Tyto závody se zobrazí daným uživatelům.



Obrázek 4: Obrazovka pro výběr závodů

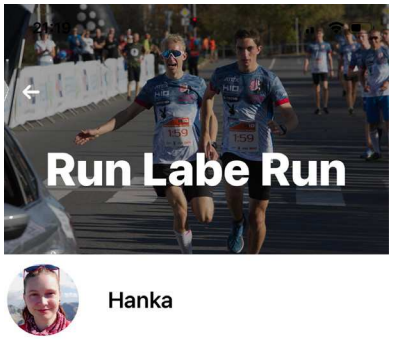
3.3.1.4 Obrazovka s vybranými závody

Obrazovka s vybranými závody je tvořena pomocí view `List` a buňky, která se vysvítí modře, pokud jsou na závod přihlášeni uživatelé. V opačném případě se uživateli otevře vyskakovací okénko, které uživatele informuje o tom, že na závod není nikdo přihlášen. Toto okénko tvořím pomocí struktury `Alert`. Závody jsou zobrazeny v elementu `List`, což také znamená, že mu můžeme přiřadit modifikátor `onDelete`. Tento modifikátor uživateli umožní přejet prstem po obrazovce a odhlásit se tak ze všech disciplín daného závodu.

Obrázek 5: Obrazovka s vybranými závody

3.3.1.5 Obrazovka se seznamem uživatelů na daném závodě

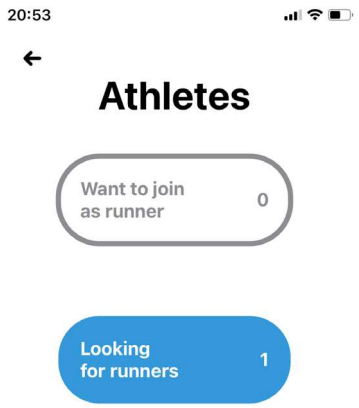
Po rozkliknutí skupiny se vypíše seznam uživatelů, kteří jsou na závod přihlášeni. Tito uživatelé jsou seřazeni podle blízkosti výkonnosti k aktuálnímu uživateli. Tato obrazovka je vytvořena pomocí view `List`.



Obrázek 6: Obrazovka s se seznamem uživatelů na daném závodě

3.2.1.6 Obrazovka s počty uživatelů v kategoriích

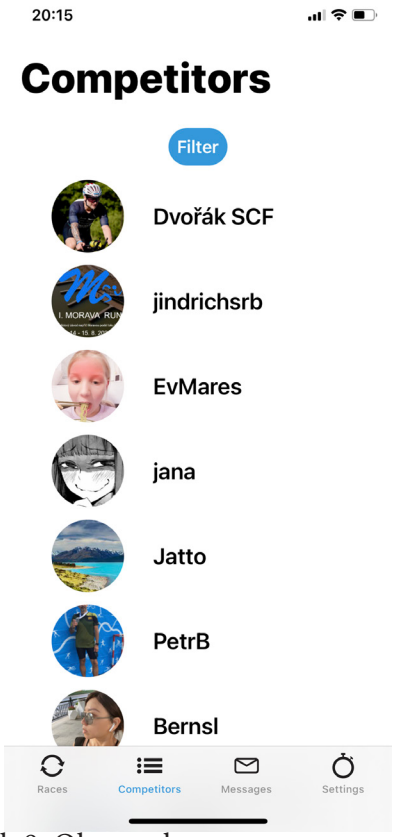
Po rozkliknutí závodu se zobrazí skupiny uživatelů roztříděné podle disciplín. Obrazovka byla opět vytvořena pomocí elementu `List` a pro buňku obrazovky jsem vytvořil samostatný view `GroupCell`.



Obrázek 7: Obrazovka s počty uživatelů v kategoriích

3.2.1.7 Obrazovka se seznamem uživatelů

Po otevření obrazovky „ Competitors“ se uživateli otevře obrazovka se všemi uživateli aplikace. Pro zrychlení načítání obrázků načítám obrázek uvnitř každé buňky těsně před jejím zobrazením. Pro správu filtrování uživatelů na této obrazovce vytvořím `ObservableObject` `CompetitorsManager`. V této třídě definuji metodu `getSortedRacers`, které mi zobrazí uživatele seřazené podle blízkosti k mému času. K zobrazení dialogu použiji roletu zobrazenou pomocí struktury `ActionSheet`. Po klepnutí na některou z buněk se uživatelé profiltrují dle disciplín.



Obrázek 8: Obrazovka se seznamem uživatelů

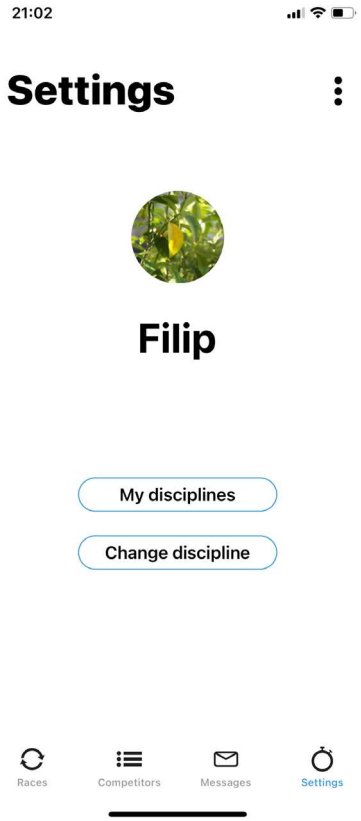
3.2.1.8 Obrazovka s chatem

Po klepnutí na uživatele se otevře chatovací obrazovka, na které může uživatel zahájit chat. Uživatel může poslat textovou zprávu nebo obrázek. I obrázek v chatu je zmenšen aby nezabíral příliš mnoho uložiště. Uživatel může snadno komunikovat detaily k danému závodu s ostatními. Po stisknutí tlačítka pro výběr obrázku se otevře roleta se seznamem obrázků. Po výběru obrázku je uživateli zobrazen jeho náhled a stisknutím tlačítka s logem šipky lze obrázek odeslat.Textové pole je tvořeno strukturou představenou v iOS 14 - `TextEditor` podporuje více řádků a je ho možné vytvořit i bez použití protokolů `UIViewRepresentable`.

Obrázek 9: Obrazovka s chatem

3.2.1.9 Obrazovka nastavení (Settings)

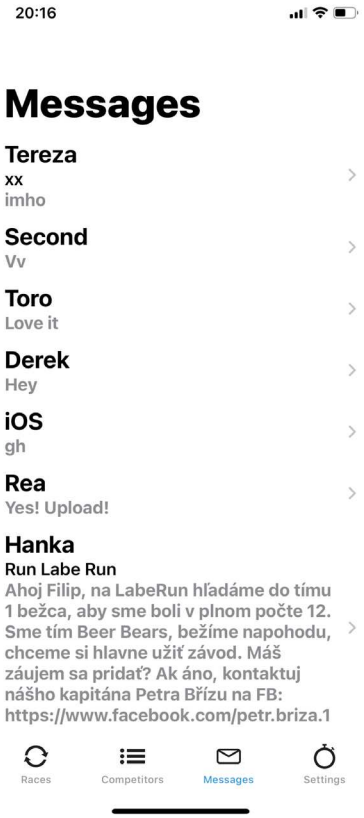
Na obrazovce Settings je zobrazen profilový obrázek uživatele a pole pro změnu přezdívky. Dále je na obrazovce zobrazeno tlačítko pro zobrazení vybraných disciplín (My disciplines). Uživateli se po otevření tohoto View zobrazí ikonky spolu s vybraným časem. Pokud se jedná o některou z disciplín, u které není uveden čas (například kajak, bruslení, paraglide nebo paddleborad) je u ikonky zobrazena ikonka zeleného checkmarku. Po kliknutí na tlačítko s názvem „ Change discipline“ se uživateli zobrazí seznam disciplín, které lze měnit. Pokud uživatel vybere disciplínu s názvem „ Swim“, „ Bike“ nebo „ Run“, zobrazí se mu okénko, ve kterém si může zvolit čas v povoleném časovém intervalu. Při použití struktury `Picker` ze SwiftUI bylo nutné zkombinovat dva elementy. Během scrolování začal scrolovat jiný sloupce, než na kterém měl uživatel prst. Proto jsem uživatelské rozhraní pro výběr času vytvořil pomocí `UIViewRepresentable`, kde jsem musel použít třídu ze staršího frameworku UIKit: `UIPickerView`. První pro výběr minut a druhý pro výběr vteřin.



Obrázek 10: Obrazovka nastavení

3.2.1.10 Obrazovka se zprávami (Messages)

Poslední podstatnou obrazovkou je obrazovka Messages, na které se zobrazují zprávy od uživatelů. Tato obrazovka je opět vytvořena pomocí elementu `List`. Tento prvek uživatelského rozhraní jsem obalil do `NavigationView` pro zajištění navigace. Uvnitř view `NavigationLink` jsem uživatele přesměroval na chatovací obrazovku.



Obrázek 11: Obrazovka se zprávami

3.2.1.11 iOS Navigace

Pro zajištění navigace jsem použil přepínání v kořenovém view projektu pomocí `ObervableObject` a v případě obrazovky Messages jsem využil `NavigationView`. Pro zajištění centrální navigace mezi obrazovkami jsem vytvořil `TabView`. Vytvořil jsem view pro každou každou z obrazovek a pomocí modifikátoru `tabItem` jsem každé z položek přiřadil název a ikonku pomocí elementů `Image` a `Text`. Ikonky jsem tvořil ve vektorovém programu Adobe Illustrator. Pokud má uživatel nepřečtenou zprávu, načtu si o této skutečnosti informaci z databáze a zobrazím u této ikonky upozornění.

3.2.1.12 Dark mode

SwiftUI podporuje dark mode (tmavý režim) automaticky. Pokud si potřebuji uživatelské rozhraní uzpůsobit podle sebe, použiji pro zjištění aktuálního režimu property wrapper `@Environment(\. colorScheme) var colorScheme`.

Poté lze opět změnit barvu textu na základě rovnosti s výčtovým typem `.dark`. V Xcode si definuji barevené schéma pro tmavý a světlý režim zvlášť, a to uvnitř složky Assets.xcassets. V kódu aplikace barvu použiji pomocí názvu například `Color("aqua")`.

3.2.1.13 Notifikace

Pro zprovoznění notifikací je nutné být členem programu Apple Developer. Pro zobrazení notifikace je potřeba využít funkci uvnitř node.js ve webovém rozhraní. Tato funkce pošle data v požadovaném fomátu a server APNS zobrazí notifikaci. Ve vývojářské konzoli si vygeneruje APNS certifikát, který poté nahraji do konzole Firebase. Nejprve si při startu aplikace zajistím potvrzení od uživatele pomocí třídy `UNUserNotificationCenter.current()`. Následně zavoláme metodu `UNUserNotificationCenter.current().requestAutorization(options: [.alert, .badge, .sound])`. Tato metoda nám vrátí informaci o tom, jestli uživatel povolil danou notifikaci či nikoli.

Node.js je open-source runtime prostředí pro spuštění Javascriptu na straně serveru. Je možné pomocí něj vytvářet backendové služby webových aplikací a jeho výhodou je, že využívá rozšířený jazyk JavaScipt. Pro zasílání notifikací je nutné zapsat funkci v node.js na webu pomocí funcke `sendToDevice(registrationToken: string | string[], payload: admin. messaging.MessagingPayload, options?: admin.messaging.MessagingOptions)` Tato funkce se dá využít na platfromách iOS, Android i web. Pomocí funkce `.onWrite((change, context) => {` si zjistím, zda proběhl zápis do entity „ Chat“ v databázi a následně si přečtu data, která uživatel poslal. Data o každé zprávě přečtu z objektu `snapshot`. Do argumentu `child` dosadím stejný název klíče, který jsem zapsal do databáze, například

```
let from = snapshot.child("from").val();
```

Z těchto argumentů poskládám payload, který zašlu na zařízení s daným tokenem. Pro otevření obrazovky po klepnutí na notifikaci ve třídě AppDelegate.swift implementuji funkci `userNotificationCenter(_ center: UNUserNotificationCenter, didReceive`

`response: UNNotificationResponse`, která vybere data z dané notifikace. Data jsou uložena v proměnné `response.notification.request.content.userInfo.` V kódu níže je ukázáno, jak přečíst data z dané notifikace.

```
let data = response.notification.request.content.userInfo

let from = data["gcm.notification.from"]!
```

Například z pole from si přečtu ID odesílatele. Pomocí následující funkce vytvořím interní notifikaci, na kterou poté zareaguji v modifikátoru `onReceive` ve SwiftUI. V tomto modifikátoru poté otevřu chatovací obrazovku.

3.3.2 Operační systém Android

Při vývoji pro Android jsem se potýkal s problémem závislostí a kompatibilitou Gradle. Častým problémem, na který jsem s tímto frameworkem narazil byla nekompatibilita tohoto systému s projektem, která bránila kompilaci. Tomuto se dalo zabránit aktualizováním Gradle a/nebo úpravou verzí některé ze závislostí. Ve Flutteru jsem pro výběr použil závislost flutter_picker. Tento plug-in jsem nainstaloval pomocí zadání závislosti do souboru pro správu závislostí `pubspec.yaml`. Následně stačilo spustit příkaz `pub.get` a tímto závislost nainstalovat. Flutter je tvořen widgety, které byly inspirovány frameworkem React. Widgety slouží k reprezentaci uživatelského rozhraní a lze je vzájemně vnořovat. Při změně stavu aplikace se změní reprezentace widgetu a framework tento widget překreslí za jiný. Stav aplikace lze změnit pomocí funkce `setState`. Po zavolání této funkce Flutter s maximální efektivitou překreslí uživatelské rozhraní. Každá aplikace ve Flutteru začíná uvnitř funkce `main`, která volá funkci `runApp`.

Funkce `runApp` nám vrací požadovaný widget.Widgety je možné do sebe libovolně zanořovat. Protože by bylo příliš nepřehledné psát celou strukturu aplikace do jednoho Widgetu, jsou často Views ve Flutteru vnořeny do rodičovského widgetu. Při tvorbě aplikace jsem cílil na to, aby byl nejvyšší widget aplikace zapsán uvnitř funkce `runApp` pouze názvem instance. Přispívá to k lepší čitelnosti kódu.

3.3.1.1 Přihlašovací obrazovka

Ve Flutteru vytvořím textové pole pomocí widgetu `TextField` a třídy TextFieldController. Tuto třídu použiji v argumentu `controller` u třídy `TextField`. V přihlašovací funkci si tyto hodnoty přečtu, uživatele přihlásím a následně jej přesměruji na úvodní stránku aplikace. Pomocí `Future<String>` vytvořím asynchronní funkci pro přihlášení a pomocí volání funkce `auth.signInWithEmailAndPassword(email: email, password: password)` uživatele přihlásím.

3.3.1.2 Registrační obrazovka

Nejprve jsem vytvořil nový widget pro registraci uživatele. Pro tvorbu textových polí jsem použil widget `TextField` společně s instancí třídy `TextEditingController`. Každý TextField má svůj `TextEditingController`, přes který lze přečíst textová data Registraci

v aplikaci pro Android zajistím pomocí funkce `await FirebaseAuth.instance.createUserWithEmailAndPassword`. Před touto funkcí jsem použil klíčové slovo `await`, abych mohl počkat na výsledek této asynchronní operace. Po úspěšné registraci je uživatel přesměrován na úvodní obrazovku aplikace se seznamem závodů. Pro přihlašovací tlačítko jsem použil třídu `TextButton` a přiřadil jsem mu jednotný styl pomocí `Outline`.

Picker se ve Flutteru vytvoří voláním konstruktoru `Picker`. Za argument `adapter` dosadím instanci třídy `PickerDataAdapter`, která obsahuje pole `pickerData`.Toto pole jsem naplnil předdefinovanými textovými řetězci v násobcích deseti. Pole z widgetu získám pomocí volání metody `picker.getSelectedValue()`. Následně pomocí třídy `Updater` zavolám metodu `updateSingleField` s hodnotami „ swim" a časem získaným z pole. První položka z pole slouží k zápisu minut, druhá položka slouží k zápisu sekund. Ve Flutteru jsem pro výběr použil závislost `flutter_picker`. Tento plug-in jsem nainstaloval pomocí zadání závislosti do souboru pro správu závislostí pubspec.yaml. Následně stačilo spustit příkaz `pub.get` a tímto závislost nainstalovat.

3.3.1.3 Obrazovka pro výběr závodu

V pravé horní části obrazovky je tlačítko na výběr závodů. Po rozkliknutí se uživateli otevře seznam dostupných závodů. Uživatel může klepnout na tlačítko „Pick", které mu otevře další widget, ve kterém se může zaregistrovat na závod. Tlačítko je vytvořeno pomocí třídy `ElevatedButton`. Pokud nemá vybraný čas dané disciplíny nebo nepotvrdil, že danou disciplínu provozuje, otevře se mu obrazovka, na které musí před výběrem závodu tuto skutečnost potvrdit. Teprve poté se jeho ID připíše k záznamu do databáze. Na této obrazovce se nachází i textové pole, přes které se dá závod filtrovat pomocí názvu. Přes další tlačítka se závody dají filtrovat i podle disciplín nebo státu, ve kterém se závod koná. Pomocí tlačítka „Close" se poté uživatel vrátí zpět na obrazovku s jeho vybranými zívody.

3.3.1.4 Obrazovka s vybranými závody

Pro obrazovku s vybranými závody jsem použil `ListView.builder`. Za argument `count` jsem dosadil počet závodů a za argument `itemBuilder` jsem pomocí closure vytvořil buňku listu. Tato buňka je vytvořena pomocí instance widgetu `ListTile`, který mi zajistí vhodné formátování.

Obrázek 12: Obrazovka s vybranými závody (Android)

3.3.1.5 Obrazovka se seznamem uživatelů na daném závodě

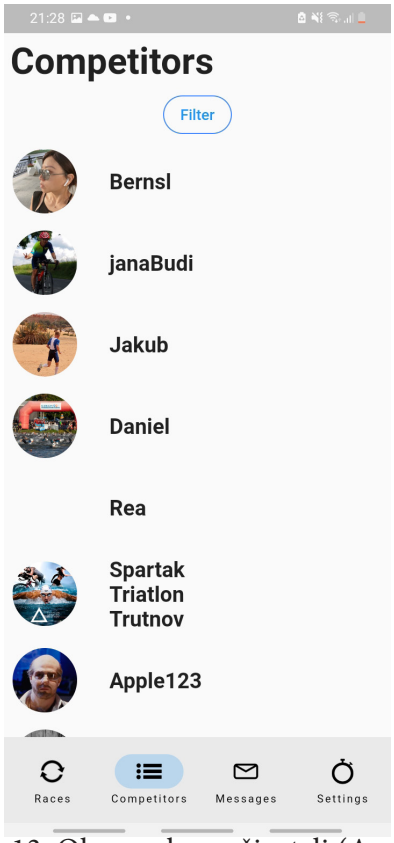
Tento widget zobrazí uživatele přihlášeného na daný závod. Obsahuje `StretchyHeader` s instancí třídy `HeaderData`, která mi zobrazí jméno závodu na elastickém pozadí. Po přejetí prstem po této obrazovce se zvětší obrázek na pozadí, což vytvoří moderní efekt.

3.3.1.6 Obrazovka s počty uživatelů v kategoriích

Pro obrazovku s počty uživatelů v daných kategoriích jsem použil widget `Column`, který mi ostatní widgety uspořádá vertikálně. Každý widget po stisknutí otevře obrazovku se seznamem uživatelů dané kategorie.

3.3.1.7 Obrazovka se seznamem všech uživatelů

Obrazovka se seznamem uživatelů byla vytvořena pomocí `ListView.builder`. Uživatelé se zde dají filtrovat podle disciplín. Roletu pro filtrování uživatelů jsem vytvořil pomocí widgetu `modalBottomSheet`.



Obrázek 13: Obrazovka s uživateli (Android)

3.3.1.8 Android Chat

Flutter sám o sobě neobsahuje knihovnu pro výběr obrázků. Tento problém se dá vyřešit instalací balíčku z webu pub.dev. K tomuto je potřeba otevřít soubor pubspec.yaml a napsat závislost image_picker. Poté ve složce projektu napíšu příkaz `pub get` anebo klepnu na tlačítko „pub get“ v textovém editoru v Android Studiu. Tímto dojde k instalaci knihoven. S potřebou externích knihoven také souvisí asi největší problém, na který jsem narazil při tvorbě aplikace. Pokud se některá knihovna změní na verzi nekompatibilní s jinou knihovou, nebude v takovém případě možné aplikaci zkompilovat a nahrát na Google Play. Hledání a nahrazení této závislosti zabralo ve spoustě případů mnoho času.

3.3.1.9 Android obrazovka nastavení (Settings)

Pole pro změnu přezdívky jsem vytvořil pomocí třídy `TextFormField`, za argument `controller` jsem dosadil instanci třídy `TextEditingController()`. V argumentu `onEditingComplete` jsem zavolal funkci pro aktualizování dat uživatele poté, co uživatel ukončí psaní. Za parametr `TextDecoration` jsem dosadil instanci třídy `InputDecoration`, kde jsem pomocí `border: InputBorder.None` odstranil barvu rámečku pole. Pomocí argumentu `hintText` jsem nastavil

popisek pole. Pomocí `TextStyle` jsem poli nastavil tučný font a pole jsem zarovnal na střed pomocí `TextAlign.center`. Pro změnu profilového obrázku uživatele jsem použil instanci třídy `ImagePicker`. Pomocí metody `pickImage` jsme vybral obrázek a uvnitř `setState` jsem ho proměnil do globální proměnné `picked`. Po nahrání nového obrázku jsem získal jeho URL a z něj jsem vykreslil nový obrázek v uživatelském rozhraní.

3.3.1.10 Obrazovka se zprávami (Messages)

Pro tvorbu obrazovky se zprávami jsem použil widget `ListView` a jeho buňku jsem vytvořil pomocí widgetu `Column` pro vertikální uspořádání. Na prvním řádku se zobrazí jméno uživatele a na druhém řádku poslední zpráva od daného uživatele.

3.3.1.11 Navigace

TabBar se ve Flutteru vytvoří využitím widgetu `BottomNavigationBar`. Za argument `unselectedItemColor` dosadím barvu nevybraného elementu. Ve funkci `onTap` zjistím index vybraného prvku pomocí hodnoty proměnné `value`. Za argument `items` dosadím pole s položkami typu `BottomNavigationBarItem`. Každá z těchto instancí do argumentu přebírá prvky `icon` a `label`. Argument `icon` slouží k předání ikonky. Ikonku do aplikace dosadím pomocí widgetu `AssetImage`. Za label „ Races" dosadím obyčejný textový řetězec.

Uvnitř TabBaru jsem vytvořil funkci `fetch`,ve které si přečtu data z položky databáze Unread. Při každém zápisu do této části databáze změním stav této obrazovky a na TabBaru překreslím uživatelské rozhraní. Pro indikaci nepřečtené zprávy jsem se rozhodl vytvořit widget `BadgeCount`. Widget `BadgeCount` využívá widget `Stack`, který obsahuje navrstvený text na červené ikonce. Navrstvení widgetů je dosaženo pomocí widgetu `Position`, kterým překryji widget s ikonkou zprávy. Navigaci mezi obrazovkami jsem řešil pomocí widgetu `MateriaPageRoute`, do kterého jsem nasadil příslušnou obrazovku.

3.3.1.12 Notifikace

Notifikace slouží pro zobrazení zpráv uživateli o tom, že mu někdo poslal zprávu ve chvíli, kdy se uživatel nacházel v aplikaci. Nejprve jsem vytvořil widget pro zobrazení notifikace `OverlayEntry`. V argumentu `builder` opět využiji widget `Position` a udám mu pozici. Do proměnné `state` vložím hodnotu `Overlay.of(context)`, což představuje vrstvu, která se nachází vizuálně nad rozhraním celé aplikace. Pomocí `state.insert` notifikaci vložím a pomocí volání `await Future.delayed(Duration(seconds: 3))` a `entry.remove()` notifikaci odstraním z obrazovky. Pro zajištění notifikací na platformě Flutter jsem využil Plug-in flutter_ local_notifications. Nainstaloval jsem jej přidáním závislosti flutter_local_notifications do pubspec. yaml. Pro práci s notifikacemi na platformě Android jsem vytvořil třídu `NotificationListener`. Definoval jsem metodu `addMessageListener`, která mi v callbacku `FirebaseMessaging.onMessage.listen` zobrazí notifikaci uživateli, když se aplikace nachází v popředí.

V uživatelském rozhraní notifikaci zobrazím pomocí metody `showOverlay`. Metoda

`showOverlay` mi notifikaci vytvoří pomocí instance třídy `OverlayEntry`. V argumentu `builder` použiji widget `Positioned`, který mi umožní nastavit si pozici pomocí argumentů `top, right a left`. Do argumentu `child` dosadím widget s vlastním obsahem uživatelského rozhraní. Výchozí bod pro umístění widgetu vytvořím pomocí `Overlay.of(context)`. Pomocí `state.insert(entry)` vložím overlay do uživatelského rozhraní. Funkcí `Future.delayed(Duration(seconds: 3))` vytvořím třívteřinovou pauzu a pomocí `entry.remove` vstup odstraním.

Uvniř třídy `overlayWrapper` vytvořím třídu `tappable`, která umožní otevřít obrazovku po stisknutí widgetu. Tohoto docílím použitím widgetu `InkWell` a metody `onTap`, kde otevřu obrazovku. Novou obrazovku lze otevřít pomocí metody `Navigator.push`, uvnitř které vytvořím cestu `MaterialPageRoute` s chatovací obrazovkou.

3.4.1 Webová aplikace

Webovou aplikaci jsem vytvořil pomocí frontendové knihovny React. React je deklarativní knihovna, sloužící k tvorbě uživatelských rozhraní. Přidání závislostí v Reactu bylo na rozdíl od Flutteru o dost jednodušší. Stejně jako v každé jiné javascriptové aplikaci stačí přidat závislost do značky `<script>` v sekci `<head>` dané stránky. Tímto způsobem jsem do projektu přidal jak samotnou verzi Relays, tak i knihovnu na tvorbu stránkování.

Rozhodl jsem se nepoužívat stávající CSS frameworky a aplikaci jsem nastyloval pomocí CSS stylů. Aplikace je zcela responzivní, což znamená, že se přizpůsobí jakékoli velikosti úhlopříčky mobilního zařízení. Responzivita byla zajištěna pomocí CSS media queries.

3.4.1.1 Přihlašovací obrazovka

Pro přihlášení na web použiju element `<input>`, který zapíšu pomocí JSX. Ve funkci `onChanged`, pomocí funkce `setState` zapíšu do stavu aplikace novou hodnotu.Tuto hodnotu následně přečtu ve funkci `login`. Zavolám v ní funkci `firebase.auth().signInWithEmailAndPassword`, která uživatele přihlásí.Tato funkce je v knihovně Firebase definována jako `Promise` a ve funkci `catch` mi umožní ošetřit případné chyby. Pokud se přihlášení podaří, uživatele přesměruji na úvodní obrazovku.

3.4.1.2 Registrační obrazovka

Ve webové verzi aplikace jsme se rozhodl pro změnu velikosti obrázků, aby nezabíraly příliš mnoho prostoru v uložišti. Podmínky, které aplikace kontroluje, jsou následující:

- e-mail musí obsahovat zavináč
- heslo musí mít délku minimálně 6 znaků
- uživatel si musí vybrat profilový obrázek
- uživatel si musí vybrat přezdívku
- všechna pole registrace jsou povinná

Kromě těchto kritérií na straně frontendu ještě kontroluji jakoukoli chybovou hlášku, kterou by mi vrátily servery Firebase. Mezi tyto chybové hlášky může patřit:

1) uživatel už je zaregistrovaný s daným e-mailem

2) e-mail je v nesprávném formátu

3) heslo je příliš krátké.

Případnou chybovou hlášku opět zobrazím v uživatelském rozhraní. Pro vypsání chybové hlášky používám červený a tučný font, aby si ji mohl uživatel jednoduše přečíst. Jako rozhraní pro výběr času jsem se rozhodl použít jednoduché textové pole vytvořené pomocí elementu `<input>` o typu `text`. V tomto případě je nutné kontrolovat správný formát textového řetězce, aby se zabránilo chybám při následném načítání z databáze. Kontroluji formát a řetězce, a také to, zda jsou sekundy i minuty v daném intervalu. Na mobilních aplikacích není potřeba kontroly formátu ani rozsahu, protože rozsah i formát jsou dány prvkem na výběr formátu. Uvnitř komponentu `<SignUp/>` jsem vytvořil funkci `resizeImage`, která obrázek po nahrání uživatelem zmenší na velikost 400 × 400 obrazových bodů.

K detekci nahrání obrázku jsem využil callback funkce `reader.onload`, kde jsem si načetl obrázek pomocí `e.target.result`. Dále jsem vytvořil element `<canvas>`, na jehož kontext jsem nakreslil nově zmenšený obrázek.

Pro kontrolu času během registrace jsem ve webové verzi rozdělil čas pomocí metody `split` na dvě části - sekundy a minuty.Pro kontrolu času jsem provedl následující kroky:

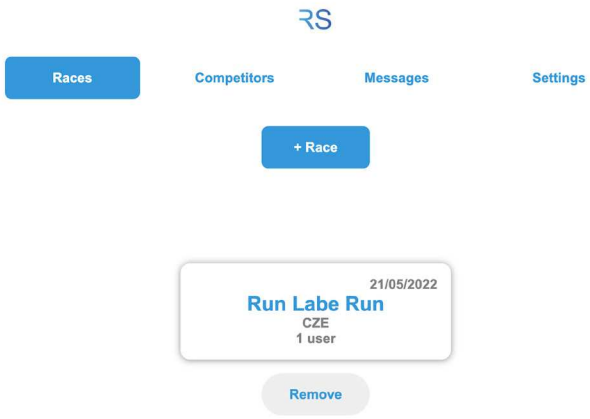
1) čas jsem rozděllil na dvojtečce na sekundy a minuty

2) zkontroloval jsem, zda jsou sekundy i minuty v požadovaném intervalu.

Pokud všechny tyto podmínky souhlasily, byl jsem schopný uživatele zaregistrovat. Během registrace jsem vytvořil funkci, která každých 500 milisekund kontroluje, zda uživatel ještě píše. Pokud uživatel již přestal psát a text nesplňuje jednu z daných podmínek, dané textové pole zčervená a uživateli se na obrazovce objeví chybová hláška.

3.4.1.3 Obrazovka pro výběr závodu

Pro tvorbu obrazovky pro výběr závodů jsem vytvořil komponent `RaceEdit`. Tento komponent načítá závody z databáze a zobrazuje je v uživatelském rozhraní. Otevře se další komponent s názvem `WhoDoYouWannaBe` s možností výběru disciplíny. Po výběru některé z disciplín se ID aktuálně přihlášeného uživatele připíše do databáze.



Obrázek 14: Úvodní obrazovka webové aplikace

3.4.1.4 Obrazovka s vybranými závody

Pro obrazovku s vybranými závody jsem vytvořil komponent `RaceSelector`. Tento otevře obrazovku s počtem přihlášených uživatelů a data předá dalšímu komponentu pomocí uložiště `localStorage`.

3.4.1.5 Obrazovka s počty uživatelů v kategoriích

Pro tuto obrazovku jsem vytvořil komponent `DisciplineSelector`, uvnitř kterého jsou tlačítka vytvořená pomocí komponentu `DisciplineButton`. Zde využívám one-way tok Reactu tak, že do `props` (argumentů) komponentu `DisciplineSelector` navážu funkci na změnu stavu, který překreslí obrazovku. Tuto funkci potom zavolám z interních komponentů `DisciplineButton`. Nenulový počet uživatelů indikuji modorou barvou. Změny barvy docílím změnou CSS třídy v komponentu.

3.4.1.6 Obrazovka se seznamem uživatelů na daném závodě

Pro tuto část aplikace jsme vytořil komponent `UserList`. Aplikace pomocí metody `getIDS` načte ID uživatelů. Z těchto ID se načtou informace o reálných uživatelích z databáze.

3.4.1.7 Obrazovka se seznamem všech uživatelů

Uvnitř komponentu pro obrazovku Competitors jsem načetl uživatele z databáze aplikace a pomocí metody `map` jsem pro každého uživatele vytvořil novou buňku. Tento komponent umožňuje taktéž filtrovat uživatele dle disciplín. Data o uživatelích nejprve uložím do uložiště `localStorage`.

Poté uživatele profiltruji dle disciplín a aktualizuji jejich seznam. React mi umožňuje pružně reagovat na změny stavu aplikace a překreslovat uživatelské rozhraní. Abych uživatele neustále informoval o tom, co se děje, vytvořil jsem během načítání komponent `<Placeholder/>` s textem „ Loading...". Po načtení uživatelů tento text zmizí. Tento princip jsem využil i na ostatních obrazovkách aplikace. Pro filtrování uživatelů jsem vytvořil komponent `<FilterButton/>`. Toto tlačítko má na sebe navázanou funkci `filterUsers`, která mi uživatele profiltruje výše zmíněným způsobem.

Zde jsem se setkal s problémem, kdy se mi nepodařilo vytvořit efekt najetí myší pomocí CSS tříd. Proto jsem vytvořil dvě funkce - `over`, která spustí se po najetí myši a `out`, která se spustí po opuštění daného prostoru myší. Uvnitř těchto dvou funkcí jsem adekvátně změnil barvu pozadí tlačítek. Tyto funkce jsme přiřadil do eventů `onMouseOver` a `onMouseOut`.

3.4.1.8 Obrazovka s chatem

Pro tvorbu chatu jsem vytvořil komponent `ChatWrapper`. Tento komponent jsem rozdělil do subkomponentů `ChatHeader` a `ChatBody`. `ChatBody` se stará o posílání zpráv a `ChatHeader` o zobrazení jména a profilového obrázku uživatele. Chat umožňuje posílat i obrázky a zmenšuje je tak, aby nezabíraly příliš mnoho místa v uložišti.

3.4.1.9 Obrazovka nastavení (Settings)

Pro načtení obrázku používám třídu `FileReader`. Metodě `onloadend` přiřadím funkci `handleFile`. Následně zavolám funkci `data.readAsText(file).` Uvnitř této funkce si načtu obrazová data argumentu pomocí `e.target.result` Následně vložím obrázek do databáze pomocí `firebase.storage().ref().put(file).` Metoda `put` vrací `promise`, a proto mohu přiřadit další funkci `then(snap => snap.ref.getDownloadURL())`, čímž získám URL adresu právě nahraného obrázku. Tuto URL adresu nahraji do databáze a aktualizuji uživatelské rozhraní.

Tato obrazovka také obsahuje textová pole pro aktualizaci časů a tlačítka pro potvrzení jednotlivých disciplín. Pro indikaci stavu vybrání disciplíny jsem vytvořil ikonky v programu Adobe Illustrator. Pro tvorbu pole na změnu přezdívky jsem využil alternativu k zápisu komponentu pomocí funkce hook. Vlastní hook lze v Reactu vytvořit pomocí definice prosté funkce. Tento hook načítá z databáze přezdívku uživatele a následně ji zobrazí v uživatelském rozhraní.

Hook jsem nazval `useNicknameHook`. Nejprve jsem pomocí `const [nick, setNick] = React.useState(null)` vytvořil proměnnou obsahující text přezdívky. Použitím funkce `React.useEffect(() => {` jsem zajistil automatické spuštění kódu pro načtení stávající přezdívky uživatele.

Přezdívku jsem přečetl z pole `Users` na pozici aktuálního ID uživatele. Hook funguje tak, že `nick` v hranaté závorce označuje aktuální hodnotu proměnné a druhá hodnota (`setNick`) značí funkci, která dokáže tuto hodnotu změnit. Jako hook jsem v této funkci použil hook na změnu stavu - `React.useState()`. Poté jsem tento hook použil v komponentu `NickFieldHook`. Pro změnu hodnoty přezdívky uživatele jsem opět vytvořil hook `React.useState()`. Uvnitř funkce

changeText přistoupím v databázi k datům právě přihlášeného uživatele a změním mu přezdívku.

Dále zavolám funkci setNick, která nám přezdívku vepíše do uživatelského rozhraní. Uvnitř uživatelského rozhraní definuji pomocí JSX element input, kde v atributu defaultValue načtu výchozí hodnotu tohoto pole. O této hodnotě rozhodnu podle toho, zda uživatel začal psát novou přezdívku. Funkci onChange zavolám v argumentu changeText.

3.4.1.10 Obrazovka se zprávami (Messages)

Tato obrazovka načítá zprávy z entity Messages. Používám funkci reduce pro roztřídění zpráv podle daných ID. Následně zobrazím první zprávu od každého uživatele a zapíšu ji do uživatelského rozhraní.

3.4.1.11 Navigace

Protože samotná knihovna React neobsahuje vlastní komponent na tvorbu routingu, využil jsem knihovnu třetí strany React Router. Navigaci jsem vytvořil pomocí komponentů <ReactRouterDOM.HashRouter> a <Link>, který v atributu to obsahuje název cesty. Samotná cesta ke komponentu je vytvořena pomocí komponentu <Route>, který do atributu component přebírá třídu komponentu uživatelského rozhraní. Pokud nyní v prohlížeči uživatel zadá relays.app/app.html#/messages, bude mu zobrazena stránka vykreslená komponentem <Messages>.

3.4.1.12 Dark mode

Někteří uživatelé dnes upřednostňují tmavý režim (dark mode) namísto světlého režimu. Ten může lépe působit na oči uživatele v noci. Ve webové verzi aplikace jsem využil uložiště localStorage. LocalStorage je lokální uložiště uvnitř webového prohlížeče, které na disk uživatele zapíše data. Do tohoto pole zapíšu hodnotu pomocí localStorage.setItem(“theme”, “dark”). Data si z localStorage přečtu pomocí localStorage.getItem(“theme”). Pokud je hodnota v localStorage „ dark”, přiřadím elementům CSS třídu pro tmavé rozhraní:

```
<div className={this.props.theme == “DARK” ? “timeSection whiteDark whiteDarkBorder” : “timeSection”}>
```

Pro barvu pozadí v tmavém režimu jsem se rozhodl nevyužívat sytou černou o hexadecimálním kódu #000, ale raději méně sytou černou (#1c1e21). Toto působí jemněji na oči uživatele. Pro přepínání mezi tmavým a světlým režimem používám ikonku měsíce, kterou jsem nakreslil ve vektorovém programu Adobe Illustrator. Po kliknutí se zapíše hodnota barveného schématu do localStorage a rovnou se překreslí barva pozadí aktuální obrazovky.

3.5 Databáze Firebase

Pro technologii serverové strany je možno použít více přístupů. Jako optimální řešení jsem zvolil knihovnu Firebase.

3.5.1 Struktura chatu v databázi

Strukturu chatu v databázi tvoří záznam vytvořený pomocí páru key-value (klíč-hodnota).

Struktura v databázi je následující:

klíč `"body"` - obsahuje zprávu

klíč "`date`" - obsahuje aktuální datum

klíč "`from`" - obsahuje ID odesílatele

klíč: "`imageurl:`" - obsahuje adresu obrázku

klíč "`rname`" - obsahuje jméno odesílatele

klíč "`rectoken`" - obsahuje token odesílatele, na který je posléze poslána notifikace

klíč "`sendertoken`" - obsahuje token uživatele

klíč "`sendurl`" - slouží k uchování adresy obrázku, pokud je součástí zprávy; v opačném případě je hodnota této hodnoty empty

klíč "`sname`" - označuje přezdívku uživatele a property ticks počet milisekund od 1/1/1970, slouží k řazení zpráv

klíč "`to`" označuje příjemce zprávy, hodnotu do tohoto pole zapíšu podle ID uživatele se kterým chatuji

3.5.2 Povolení registrace

Pro povolení registrace jsem se přepnul na záložku Sign-in method, klepnul na ikonku tužky, přepnul přepínač na hodnotu „Enable" a stisknul tlačítko OK. V konzoli jsem otevřel položku Realtime Database, a povolil jsem ji. V záložce Rules jsem zapsal pravidla pro přístup k databázi a uložil je. Tato pravidla umožní neautentizovaný zápis a čtení dat pro účely testování.

Otevřel jsem projekt Relays.xcworkspace, klepnul jsem pravým tlačítkem na soubor .GoogleServicesInfo.plist a otevřel ho jako zdrojový kód. Pod klíč IS_SIGNIN_ENABLED jsem přidal klíč DATABASE_URL s URL adresou, kterou jsem našel v konzoli. Jedná se o hodnotu <string>https://idProjektu-idDatabaze-default-rtdb.firebaseio.com/</string>. Pomocí této URL adresy bude probíhat komunikace aplikace s databází.

3.5.3 Práce s databází iOS

Při práci na aplikaci jsem dbal na kvalitu oddělení logiky aplikace od daných view. Proto jsem se rozhodl vytvořit velké množství tříd, které se starají o registraci, přihlášení, načtení uživatelů nebo

přihlášení na daný závod.

3.5.3.1 Joiner

Tato třída umožňuje přidat uživatele na daný závod. V této třídě jsem definoval metodu `joinOrRemoveIfAlreadyThereAs`, která se postará o odstranění/zapsání uživatele do příslušného pole v databázi. Kvůli efektivitě kódu je k záznamu závodu v databázi přistoupeno pomocí primárního klíče - ID závodu. Třída si z databáze přečte aktuální seznam uživatelů a odstraní z něj ID aktuálně přihlášeného uživatele. Poté data zapíše zpět do pole `users` v entitě Races v databázi.

3.5.3.2 SimpleUserGetter

Třída `SimpleUserGetter` slouží na načítání uživatelů na daném závodě. Tato obsahuje metodu asynchronní `getSimple`, která na základě disciplíny využije služby třídy `UserGrouped`.Všechny ID uživatelů jsou načteny pomocí metody `allIDSAtSingleGroup`. Tato metoda je definována uvnitř třídy `UseGrouped`. Následně si načteme data o uživatelích s těmito ID z položky databáze „Users“. Tato data dosadíme do pole s položkami datového typu `User` a toto pole vrátíme. Při použití asynchronní metody vrátíme tato data pomocí volání bloku `completion`.

3.5.3.3 MessagesGrouper

`MessagesGrouper` je třída pro načítání zpráv od uživatelů a jejich třídění. Třída si nejprve načte zprávy od všech uživatelů a poté je pomocí struktury `Dictionary` s konstruktorem `grouping(by:)` roztřídí podle ID odesílatele. Toto pole je následně vráceno do view a zobrazeno uživateli v elementu `List`.V buňce listu, pro který jsem vytvořil vlastní view, je vrácena přezdívka společně s poslední zprávou uživatele.

3.5.3.4 Třída UserFetcher

Pro čtení uživatelů z databáze jsem vytvořil třídu UserFetcher. Pro vrácení načtených dat jsem využil objekt `promise`, který reprezentuje dokončení nebo selhání asynchronní operace. `Promise` je objekt, ke kterému je možné přiřadit callbacky. Odpadá tak nutnost vkládání callbacků přímo do funkce.

`Promise` vrátí požadovanou hodnotu pomocí volání funkce `resolve`. Pokud při běhu dojde k chybě, je možné zavolat funkci `reject`. Hodnota v argumentu této funkce bude předána do výstupové funkce `then`. Do každé `promise` lze funkcí `then` přidat několik funkcí `then`. Tomuto způsobu zápisu kódu se říká chaining. Během této operace je často nutné zachytit chybu. Tohoto docílíme pomocí klauzule `catch`. Pokud na konec řetězce připíšeme další klauzuli `then`, tak operace proběhne bez ohledu na to, zda operace před ní skončila s chybou. Uvnitř třídy `July2021Fetcher` vytvořím funkci `getIDs` s argumenty `databaseName` a `discipline`.

Uvnitř této funkce vrátím funkci `Promise((resolve, reject) => {...})` s argumenty `resolve` a `reject`. Následně vytvořím funkci `getUsersFromIDs`, která mi z těchto ID načte data o daných uživatelích. Tyto funkce využívám u zobrazení účastníků jednotlivých

závodů nebo všech uživatelů v aplikaci uvnitř komponentu `CompetitorsView`.

3.5.3.5 MeFetcher

Skrze stejnou techniku přistoupím k načítání dat o aktuálním uživateli.Tuto třídu používám jak na obrazovkách s výběrem závodů pro relativní porovnání výkonnosti účastníků, tak i na obrazovce se změnou nastavení `SettingsView`.

3.5.3.6 Třída Updater

Třída Updater slouží ke změně hodnot daného uživatele v databázi. Dart podporuje asnychronní programování pomocí klíčového slova `await`, po jehož použití funkce čeká na výsledek asynchronního volání. V této třídě se slovo `async` používá pro zjištění hodnoty `Auth().auth.currentUser`, která obsahuje informace o aktuálně přihlášeném uživateli.

3.5.4 Práce s databází - Android a web

Ekvivalentní třídy jsem implementoval na Androidu pomocí `knihovny dart:async` a na webu pomocí objektu `Promise`.

3.6 Zveřejnění aplikace

Nejprve jsem aplikaci Relays zveřejnil na App Store pro platformu iOS a až poté jsem se rozhodl pracovat na její verzi pro platformy Android a web a její následné zveřejnění na Google Play a adrese www.relays.app.

3.6.1 Publikace v App Store

Proces zveřejnění aplikace zahrnuje několik kroků. Aplikaci jsem archivoval pomocí volby Product -> Archive uvnitř Xcode a následně ji nahrál do AppStore Connect. App Store Connect je služba, pomocí které můžeme nahrát aplikaci na App Store. Po přihlášení na stránku appstoreconnect.apple.com můžeme vytvořit nový záznam a aplikaci. Stiskneme tlačítko + vlevo nahoře a vybereme možnost App. Vyplníme BundleID, které nalezneme uvnitř Xcode v panelu Settings. Dále klepneme na tlačítko Create.

V konzoli dále vyplníme popisek aplikace, screenshoty, osobní informace a informace o ceně a dostupnosti. Po vyplnění všech možností klepneme na tlačítko Save. Každá aplikace musí před zveřejněním projít přísnou kontrolou Apple Review. Aplikace prochází automatickou i manuální kontrolou. Stav zveřejnění aplikace můžeme sledovat na webové stránce nebo v aplikaci AppStore Connect. Po nahrání aplikace se dostane do stavu “Waiting for review“. Pokud jsou během Review nalezeneny nějaké problémy, je stav aplikace změněn na “Rejected”. V tomto případě musíme chyby opravit nebo App Review teamu vysvětlit důvod našich akcí. Následně aplikaci odešleme k dalšímu Review. Pokud v tomto případě nejsou nalezeny problémy, aplikace přejde do stavu “Ready for sale“ a během několika minut se stane dostupnou na AppStore.

Pro testovací účely je možné použít i službu TestFlight. TestFlight funguje tak, že si uživatel

stáhne aplikaci TestFlight z AppStore a otevře pozvánku, která mu přijde přes e-mail. TestFlight můžeme nastavit v sekci TestFlight na AppStore Connect.

3.6.2 Publikace v Google Play

Na Google Play je kontrola aplikace méně přísná než na AppStore. Na rozdíl od AppStore neprobíhá důkladná kontrola a aplikace je v případě schválení zařazena do prodeje. Google Play Review proces využívá ve větší míře automatické schvalování. U určitých kategorií (například u aplikací zaměřených na děti) se do review zapojí lidský faktor. [7][8]

3.6.3 Publikace na webu

Pro nasazení webové verze do provozu jsem se rozhodl pro hosting od firmy GoDaddy. Soubory webové stránky jsem jednoduše nahrál do webového rozhraní a zveřejnil je na doméně relays.app. Tato doména byla taktéž zakoupena od společnosti Godaddy.

3.6.3.1 Uspořádání webového projektu

Při programování projektu jsem dbal na čistotu kódu a kód webové stránky jsem rozdělil na komponenty a třídy pro práci s databází. Ve struktuře webu jsem použil plochou strukturu sémanticky pojmenovaných složek.

3.6.3.2 Grafický návrh obrazovek

Protože jsem celý projekt vytvářel samostatně, nepoužil jsem jsem klasické wireframy, ale rovnou jsem začal s grafickým návrhem obrazovek. K návrhu obrazovek jsem využil aktuální verzi aplikace Adobe Photoshop. Pomocí tohoto grafického softwaru jsem navrhl snímky pro všechny hlavní obrazovky aplikace.

- Messages - Zde jsem se zaměřil na jednoduchý návrh seznamu zpráv se jménem uživatele a poslední zprávy, kterou uživatel napsal.
- Competitors - Pro obrazovku se seznamem uživatelů jsem zvolil návrh se seznamem buňek, které obsahují přezdívku a fotku uživatele.
- Users - pro obrazovku se seznamem uživatelů na daném závodě jsem vytvořil podobný styl jako u obrazovky Competitors.
- Obrazovka se seznamem vybraných závodů - pro tuto obrazovku jsem vytvořil jednoduchý design, kde jsem modře zvýraznil závody, které jsou již obsazeny.
- Obrazovka pro výběr závodů - na této obrazovce jsem jednoduše ukázal všechny závody v databázi společně s vyhledávacím polem. Nachází se zde i tlačítko pro zobrazení rolety s kategoriemi filtrů.
- Chatovací obrazovka - Tato obrazovka obsahuje přehledný seznam zpráv od daného uživatele. Obsahuje jeho profilový obrázek a přezdívku. Tlačítko pro odeslání zprávy a výběr obrázku se nachází vedle textového pole.

ZÁVĚR

Cílem bakalářské práce byl vývoj mobilní aplikace Relays pro platformy iOS, Android a web. Aplikace Relays vyhledává partnery pro štafetové závody, které se konají na celém světě. Pro všechny tři platformy je možno použít multiplatformní framework Flutter. Nejprve jsem však aplikaci vyvinul pro iOS ve SwiftUI, protože v době vývoje se jednalo o novou verzi tohoto frameworku. Po jejím zveřejnění na App Store jsem se rozhodl ji rozšířit na zbývající dvě platformy. Pro implementaci aplikace na platformě Android jsem využil deklarativní framework Flutter, protože jsem nechtěl k vývoji použít stávající nativní imperativní SDK firmy Google. Pro webovou verzi aplikace jsem použil knihovnu React, jelikož mám zkušenosti s programovacím jazykem JavaScript.

Práce se skládá ze dvou hlavních částí - teoretické a praktické. V teoretické části jsou popsány možnosti vývoje aplikace pro jednotlivé platformy z hlediska knihoven a jazyků s jejich jednolivými vlastnostmi. Pro technologii serverové strany je možno použít více přístupů, které jsou blíže popsány v teoretické části práce. Jako optimální řešení jsem zvolil knihovnu Firebase. Závěrem teoretické části byly porovnány výhody a nevýhody vývoje pro jednotlivé platformy z různých hledisek.

V praktické části byly identifikovány funkční a nefunkční požadavky pro aplikaci Relays na všech platformách. Také byl navržen vzhled obrazovek a uživatelské rozhraní odpovídající výše zmíněným požadavkům.

V posledním kroku praktické části byla aplikace implementována pro všechny tři platformy, a to ve SwiftUI pro platformu iOS, ve Flutteru pro platformu Android a v Reactu pro web. Mobilní aplikace Relays je zveřejněna na App Store a Google Play pod názvem Relays: Find a relay partner a její webová verze je umístěna na adrese www.relays.app.

V současné době obsahuje několik desítek českých i světových závodů a používá ji několik desítek uživatelů.

SEZNAM POUŽITÉ LITERATURY

[1] ZACCAGNINO, Carmine. Programming Flutter: Native, Cross-Platform Apps the Easy Way. 1. New York: Pragmatic Bookshelf publishing, 2020. ISBN 9781680506952.

[2] LACKO. Luboslav. Vývoj aplikaci pro iOS. Brno: Computer Press, 2018. ISBN 978-80-251-4942-3.

[3] Google Inc. [online dokumentace].©2022 [cit. 30.01.2022] Dostupné z: https://docs.flutter.dev

[4] Apple Inc. [online dokumentace] ©2022 [cit.30.01.2022] Dostupné z: https://developer.apple.com/documentation/swiftui/

[5] Apple Inc. [online dokumentace] ©2022 [cit.30.01.2022] Dostupné z: https://developer.apple.com/documentation/swift/

[6] Apple Inc. [online dokumentace] ©2022 [cit.30.01.2022] Dostupné z: https://developer.apple.com/documentation/uikit/

[7] Google Inc. [online dokumentace]. ©2022 [cit. 30.01.2022] Dostupné z: https://docs.flutter.dev/deployment/android

[8] Google Inc. [online dokumentace]. ©2022 [cit. 30.01.2022] Dostupné z: https://developer.android.com/distribute/best-practices/launch/launch-checklist

[9] Selectiv [online]. ©2021 [cit. 30.01.2022] Dostupné z: https://selectivv.com/en/mobile-marketing-2018-20-statistics-that-you-should-know/

[10] Statista [online]. ©2021 [cit. 30.01.2022] Dostupné z: https://www.statista.com/statistics/289418/number-of-available-apps-in-the-google-play-store-quarter/

[11] Mozilla [online]. ©2021 [cit. 30.01.2022] Dostupné z: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Using_promises

[12] Google [online]. ©2021 [cit. 30.01.2022] Dostupné z: https://firebase.google.com/docs

[13] MCGREGOR, Duncan, PRYCE Nat., Java to Kotlin, O'Reilly, Media, Inc., 2021, ISBN-9781492082279

[14] Google Inc. [online dokumentace]. ©2022 [cit. 30.01.2022] Dostupné z: https://developer.android.com/jetpack/compose/documentation

[15] Google Inc. [online dokumentace]. ©2022 [cit. 30.01.2022] Dostupné z: https://dart.dev/guides

[16] Oracle, Inc. [online dokumentace]. ©2022 [cit. 30.01.2022] Dostupné z: https://dev.mysql.com/doc/

[17] PHP Group, [online dokumentace]. ©2022 [cit. 30.01.2022] Dostupné z: https://www.php.net/docs.php

[18] OpenJS Foundation [online dokumentace]. ©2021 [cit. 30.01.2022] Dostupné z: https://

nodejs.org/en/

[19] Google Inc. [online dokumentace]. ©2022 [cit. 30.01.2022] Dostupné z: https://angular.io/

[20] Vue.js [online dokumentace]. ©2022 [cit. 30.01.2022] Dostupné z: https://vuejs.org/guide/introduction.html

[21] Microsoft Inc. [online dokumentace]. ©2022 [cit. 30.01.2022] Dostupné z: https://docs.microsoft.com/en-us/aspnet/core/blazor/?view=aspnetcore-6.0

[22] React.js [online dokumentace]. ©2022 [cit. 30.01.2022] Dostupné z: https://reactjs.org/

[23] Zaccagnino Carmine Programming Flutter: Native, Cross-Platform Apps the Easy Way (The Pragmatic Programmers) 1st Edition, Kindle Edition. ©2022, The Pragmatic Programmers, 2020. ISBN 978-1680506952

[24] Mozilla [online]. ©2021 [cit. 30.01.2022] Dostupné z: https://developer.mozilla.org/en-US/docs/Web/JavaScript/

[25] Nativní, hybridní nebo webová aplikace? Která je nejlepší? — Médiář. Médiář — Média, marketing, maloobchod [online]. Copyright © News Media 2011 [cit. 30.01.2022]. Dostupné z: https://www.mediar.cz/nativni-hybridni-nebo-webova-aplikace-ktera-je-nejlepsi/

[26] Android | Definition, History, & Facts | Britannica. Encyclopedia Britannica | Britannica [online]. Copyright © Alan Ward [cit. 30.01.2022]. Dostupné z: https://www.britannica.com/technology/Android-operating-system

[27] Google - YouTube. YouTube [online]. Copyright © 2022 Google LLC [cit. 30.01.2022]. Dostupné z: https://www.youtube.com/c/google

[28] Front end vs. Back end - jaký je mezi nimi rozdíl? - Blog - ApiTree. ApiTree - Vývoj, konzultace, design [online]. Copyright © 2022 Api Tree [cit. 30.01.2022]. Dostupné z: https://www.apitree.cz/blog/front-end-vs-back-end-jaky-je-mezi-nimi-rozdil

[29] PHP Usage Statistics. BuiltWith Web Technology Usage Statistics [online]. Copyright © 2022 BuiltWith, [cit. 30.01.2022] Dostupné z: https://trends.builtwith.com/framework/PHP

[30] Ionic Article: Ionic vs Flutter: Best Platform for Hybrid App Development. Ionic: Enterprise App Development & Delivery Platform [online]. Copyright © [cit. 30.01.2022]. Dostupné z: https://ionic.io/resources/articles/ionic-vs-flutter-comparison-guide

[31] React Native · Learn once, write anywhere. React Native · Learn once, write anywhere [online]. Copyright © 2022 Meta Platforms, Inc. [cit. 30.01.2022]. Dostupné z: https://reactnative.dev/

[32] Swift programming language: Concurrency [online]. Dostupné z: https://docs.swift.org/swift-book/LanguageGuide/Concurrency.html Copyright ©2022 [cit.30.01.2022].

[33] Objective-C to Swift - Making the Transition. CODE Online [online]. Copyright ©2021 [cit.30.01.2022] Dostupné z: https://www.codemag.com/article/1611051/Moving-Forward-

The-Transition-from-Objective-C-to-Swift

[34] GAUCHAT. JD. SwiftUI for Masterminds, 2022 Apple Books, ISBN 9780991817887

[35] SwiftUI wishlist for WWDC21 | Swift with Majid. Home | Swift with Majid [online]. Copyright © [cit. 30.01.2022] Dostupné z: https://swiftwithmajid.com/2021/05/26/swiftui-wishlist-for-wwdc21/

[36] Pros and Cons of Java Cloudflare [online]. Copyright © [cit. 30.01.2022]. Dostupné z: https://techvidvan.com/tutorials/pros-and-cons-of-java/

[37] What are The Benefits of Kotlin. Kotlin, a new programming language, has… | by Sagara Technology Idea Lab | Medium. Sagara Technology Idea Lab – Medium [online]. Copyright © [cit. 30.01.2022]. Dostupné z: https://sagaratechnology.medium.com/what-are-the-benefits-of-kotlin-d7fdcd1cfc0#:~:text=The%20business%20benefits%20of%20switching,of%20code%20compared%20to%20Java.

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM PŘÍLOH